UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

Rande Arievilo Moreira

Segurança Adaptativa em Aplicações Coletivas Cientes de Contexto

Belo Horizonte
2011

Rande Arievilo Moreira

Segurança Adaptativa em Aplicações Coletivas Cientes de Contexto

Versão Final

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Orientador: Antônio Alfredo Ferreira Loureiro

Belo Horizonte
2011

Moreira, Rande Arievilo.

M838s Segurança adaptativa em aplicações coletivas cientes de contexto [recurso eletrônico] / Rande Arievilo Moreira. – 2011.

1 recurso online (52 f. il., color.) : pdf.

Orientador: Antônio Alfredo Ferreira Loureiro.

Dissertação (mestrado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação.
Referências: f. 51-52.

1. Computação – Teses. 2. Redes de computadores – Medidas de segurança – Teses. 3. Redes de sensores sem fio – Teses. I. Loureiro, Antônio Alfredo Ferreira. II. Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação. III. Título.

CDU 519.6*43(043)

Ficha catalográfica elaborada pela bibliotecária Irenquer Vismeg Lucas Cruz
CRB 6/819 - Universidade Federal de Minas Gerais – ICEx

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Segurança adaptativa em aplicações coletivas cientes de contexto

RANDE ARIEVILO MOREIRA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. ANTONIO ALFREDO FERREIRA LOUREIRO - Orientador
Departamento de Ciência da Computação - UFMG

PROFA. RAQUEL APARECIDA DE FREITAS MINI
Departamento de Ciência da Computação - PUC-MG

PROF. RICARDO AUGUSTO RABELO OLIVEIRA
Departamento de Computação - UFOP

Belo Horizonte, 28 de fevereiro de 2011.

Dedico este trabalho àqueles que sempre me deram a oportunidade de estudar e o incentivo para ir cada vez mais longe: meus pais.

Acknowledgments

Agradeço a todos que de alguma forma estiveram envolvidos e partilharam comigo muitos momentos ao longo desses dois anos de trabalho. Em primeiro lugar, agradeço aos meus pais, que sempre deram todo o suporte para a realização dos meus sonhos. Ao meu irmão, agradeço por não ter me atrapalhado nos estudos, como nos tempos de criança. A todos os demais familiares, agradeço pelo carinho e pela confiança que sempre depositaram em mim.

Em segundo lugar, gostaria de agradecer a todos os meus amigos. Aos companheiros de trabalho na Adok, pelo ambiente descontraído. Aos chefes, pela compreensão, pela flexibilidade e pela amizade. Aos amigos de faculdade, por compartilharem comigo as dificuldades do dia-a-dia acadêmico. Agradeço especialmente à minha melhor amiga Maria Luíza, que esteve comigo desde os tempos de graduação e sempre esteve lá para me ouvir.

Por fim, mas não menos importante, àquela que apareceu na minha vida nos últimos meses e se tornou uma das pessoas mais importantes. Àquela que soube me dar a motivação necessária na reta final deste trabalho. Jordana, te amo.

"*Acredito que errado é aquele que fala correto e não vive o que diz*"
(O Teatro Mágico)

Resumo

As redes sem fio estão cada vez mais disponíveis em diferentes ambientes. Os avanços recentes em nanotecnologia tornam possível implantar sistemas computacionais em objetos cada vez menores, fazendo com que o nosso dia-a-dia seja cercado por "seres" com alguma capacidade de processamento.

Futuramente, existirão ambientes compostos por objetos inteligentes – inclusive com capacidade de se adaptarem ao meio em que se encontram – comunicando-se entre si e, possivelmente, executando alguma tarefa de forma colaborativa. Entretanto, com mais dispositivos e aplicações, maior o tráfego de informação observado, o que deixa evidente que é necessário prover algum tipo de segurança nessas situações. Seja restrição de dados, controle de acesso ou mesmo garantia de alta disponibilidade.

Existem vários fatores que podem influenciar na escolha dos algoritmos utilizados para garantir a segurança: requisitos de QoS, capacidade de *hardware*, condições da rede, etc. Cada uma dessas características influencia em maior ou menor grau na política de segurança a ser adotada, e assim, percebe-se que um conjunto estático de algoritmos não é a melhor forma de garantir a segurança de uma comunicação.

Uma solução mais intuitiva para determinação da segurança nessa situação é utilizar um mecanismo que seja capaz de adaptar e adequar a escolha dos algoritmos de acordo com o contexto em que a aplicação se encontra. Utilizando como base o *Middleware de Segurança Adaptativo para Computação Móvel*, são propostas diversas melhorias à implementação inicial e também uma nova abordagem considerando a expansão para comunicação em grupo – não suportada inicialmente pelo *middleware*, além de mecanismos para envio de mensagens ordenadas e anônimas.

Palavras-chave: segurança; redes sem fio; aplicações cientes de contexto.

Abstract

Wireless networks are available in different environments. Advances in nanotechnology make it possible to deploy computer systems in very small objects, surrounding us with "beings" with some processing capacity.

In future, we will have several environments composed by smart objects – including the ability of adapting to the environment – comunicating and, possibly, executing some task in a collaborative way. Otherwise, more devices and applications brings more information traffic and becomes evident the needs for some security. Being it data restriction, access control or high availability.

There are many factors that influence which algorithms should be chosen to warrant security: QoS requirement, hardware capacity, network conditions, etc. Each one of this features can influence the security policy to be adopted, and thus, a static set of algorithms is not the best way to warrant security in communication.

A more intuitive solution to determine security in this situation is using a mechanism that is able to adapt the security algorithms choice according to the context in which the application is. Using as a start point the *Middleware de Segurança Adaptativo para Computação Móvel*, we propose several improvements to the original implementation and also a new approach considering group communication – not supported originally by the middleware, besides of mechanisms to send ordered and anonymous messages.

Keywords: security; wireless networks; context aware applications.

List of Algorithms

Sumário

Capítulo 1

Introdução

1.1 Contextualização

As redes sem fio estão cada vez mais disponíveis em diferentes ambientes, fornecendo aos usuários acesso rápido e fácil a serviços e aplicações em qualquer lugar e a qualquer momento. Os avanços recentes em nanotecnologia tornam possível implantar sistemas computacionais em objetos cada vez menores, fazendo com que o nosso dia-a-dia seja cercado por "seres" com alguma capacidade de processamento. As redes sem fio, apesar das limitações de largura de banda algumas vezes encontradas, vem evoluindo, como pode-se perceber com o lançamento de novos padrões de comunicação na área de telefonia móvel (4G) Rumney [2008]. À medida que a tecnologia de redes sem fio avança, torna-se também mais popular. A comScore[1], revelou que, em dezembro de 2009 nos EUA, 234 milhões de pessoas com mais de 13 anos utilizavam dispositivos móveis, sendo que cerca de 43% utilizam esses dispositivos para realizar algum acesso à internet.

A quantidade de aplicações desenvolvidas para dispositivos móveis também surpreende. Em parte, esse nicho foi impulsionado pela **Apple** através da **AppleStore**, onde estão à venda milhares de aplicativos para *IPhone*. Mais recentemente, com o lançamento do sistema operacional para dispositivos móveis da **Google** – o Android [2] – e posteriormente do aparelho Google Nexus One, esse nicho tem se tornado cada vez mais interessante, tanto para desenvolvedores, quanto para usuários.

Futuramente, existirão ambientes compostos por objetos inteligentes – inclusive com capacidade de se adaptarem ao meio em que se encontram – comunicando-se entre si e, possivelmente, executando alguma tarefa de forma colaborativa. Entretanto, com mais dispositivos e aplicações, maior o tráfego de informação observado, e fica muito evidente que é necessário prover algum tipo de segurança nessas situações. Seja restrição de dados, controle de acesso ou mesmo garantia de alta disponibilidade.

[1] http://www.comscore.com/, último acesso em 17 de janeiro de 2011
[2] http://android.com, último acesso em 21 de janeiro de 2011

1.2 Objetivos e Motivação

As aplicações cientes de contexto têm sido mais amplamente exploradas na última década. A computação ciente de contexto caracteriza-se principalmente por considerar métricas do ambiente no qual o usuário está inserido para adaptar o serviço oferecido de acordo com as condições do meio. Uma definição mais detalhada sobre computação ciente de contexto pode ser encontrada na seção 2.2. Por se tratar de um tema recente, existem vários desafios de pesquisa ainda em aberto, possibilitando o desenvolvimento de novos trabalhos. Dentre as áreas possíveis, será explorado neste trabalho a segurança.

Quando considera-se um ambiente com aplicações cientes de contexto, tem-se uma situação bastante complexa para ser analisada: cada aplicação possui seus próprios requisitos de QoS; cada uma delas pode ser executada em dispositivos com características muito distintas; por trás da aplicação, pode-se ter usuários com diferentes perfis (modelo de mobilidade, por exemplo); por fim, tem-se o fator tempo, uma vez que várias das características citadas podem variar ao longo do tempo. Como essas características influenciam em maior ou menor grau a política de segurança a ser adotada, percebe-se que implantá-la não se limita simplesmente a incorporar um conjunto estático de algoritmos e utilizá-los na comunicação. Uma solução mais intuitiva para determinação da segurança nessa situação é utilizar um mecanismo que seja capaz de adaptar e adequar a escolha dos algoritmos de segurança de acordo com o contexto em que a aplicação se encontra.

Além disso, é crescente a interação de pessoas utilizando a Internet, especialmente com o advento das redes sociais. Isso traz a necessidade de se pensar em uma solução de segurança que, além de adaptativa, seja capaz de atender grupos de usuário, não apenas uma solução do ponto de vista individual.

Ainda no âmbito da Internet, percebe-se também que as pessoas comunicam-se de forma anônima, seja nas redes sociais, fóruns ou jogos *on-line*, através do uso de pseudônimos. Expandindo esse cenário para a comunicação em grupo, de forma mais abrangente, há situações em que é necessário (ou desejável) que um indivíduo seja capaz de enviar uma mensagem anônima para um grupo de usuários. Apenas como ilustração, imagine o cenário em que um usuário chega ao *shopping center* e deseja saber quais as lojas que vendem sapatos da marca Y. Para isso, ele enviaria uma mensagem anônima para todas as lojas (grupo) e receberia (mesmo sem ser identificado) uma resposta à sua solicitação.

O presente trabalho tem por objetivo instanciar um mecanismo de segurança adaptativo que considere o contexto para tomar decisões relativas à segurança das aplicações e dos dados por elas utilizados. Isso significa que, em um ambiente com várias aplicações cientes de contexto, os dados de contexto identificados serão utilizados de forma a definir qual a melhor política de segurança que pode ou deve ser utilizada em um determinado

momento. Além de ser adaptativo, deseja-se que o mecanismo utilizado suporte a comunicação em grupo e também a comunicação anônima.

Utilizaremos como referência o *Middleware de Segurança Adaptativo para Computação Móvel* Rocha [2007], que basicamente é um mecanismo de segurança adaptativo, ciente de contexto, mas restrito à comunicação par-a-par. Além dessa restrição, o *middleware* apresenta alguns problemas arquiteturais, o que acarreta em baixa manutenibilidade e baixa extensilidade. Isso dificulta o desenvolvimento de novas soluções ou mesmo de melhorias dos métodos implementados originalmente pelo *middleware*. Por exemplo, o *middleware* possui um método para escolha da melhor política de segurança de acordo com o contexto. No entanto, é difícil realizar alguma modificação nesse método sem afetar o restante do funcionamento.

1.3 Contribuição

A solução apresentada ao longo deste trabalho será capaz, entre outras coisas, de:

1. Escolher políticas de segurança para grupos de forma adaptativa;
2. Realizar comunicação anônima entre um usuário não identificado e um grupo de usuários.

Essas duas contribuições são importantes dentro do cenário atual de comunicação e interação entre pessoas utilizando redes de computadores, como já discutido na seção anterior.

Além disso, o trabalho será desenvolvido na plataforma móvel Android. Dessa forma, uma outra contribuição será a disponibilização de uma solução de segurança que pode ser diretamente incorporada por aplicações também desenvolvidas nessa plataforma. Isso facilitará o desenvolvimento de aplicações, que poderão se comunicar de forma segura, sem a necessidade de que cada uma delas implemente seu próprio mecanismo de segurança.

1.4 Organização do Texto

O restante do texto está organizado da seguinte forma: o capítulo 2 apresenta algumas definições necessárias a um melhor entendimento do trabalho proposto e também a

análise de alguns trabalhos relacionados. Os modelos de rede e de comunicação em grupo considerados são abordados no capítulo 3. No capítulo 4 serão apresentados detalhes da solução proposta, incluindo arquitetura, configuração e formas de conexão. Os resultados obtidos serão discutidos no capítulo 5. Por fim, no capítulo 6 são apresentadas as conclusões e sugestões de trabalhos futuros.

Capítulo 2

Fundamentos teóricos

A fim de padronizar conceitos que serão utilizados ao longo do texto, serão apresentadas, a seguir, algumas definições de temas relacionados ao trabalho proposto. Na seção 2.1, será abordado o conceito de segurança e algumas de suas principais características. A computação ciente de contexto será discutida na seção 2.2. A definição de *middleware* está presente na seção 2.3. Por fim, vários tipos de comunicação em redes serão apresentados na seção 2.4.

2.1 Segurança

A comunicação entre indivíduos envolve invariavelmente a troca de informações, que podem ou não ser sigilosas. Se a informação existe, mas não é divulgada (*e.g.* está fisicamente registrada em um meio ao qual ninguém tem acesso), oferecer segurança é uma tarefa mais simples. A partir do momento que essa informação torna-se visível para outras pessoas, garantir a segurança dos dados passar a ser uma tarefa muito mais complexa. No contexto de redes sem fio, a informação trafega em um meio (ar) onde pode ser captada facilmente, por isso o interesse em oferecer segurança nesse ambiente.

A tríade CIA (*Confidentiality, Integrity and Availability*) – Confidencialidade, Integridade e Disponibilidade – representa os principais atributos que, atualmente, orientam a análise, o planejamento e a implementação da segurança para um determinado grupo de informações que se deseja proteger. Outro atributo importante é a autenticidade. Com a evolução do comércio eletrônico e da sociedade da informação, a privacidade tornou-se uma grande preocupação.

O significado de cada um desses quatro atributos básicos é:

- **Confidencialidade**: propriedade que limita o acesso à informação tão somente às entidades legítimas, ou seja, àquelas autorizadas pelo proprietário da informação.
- **Integridade**: propriedade que garante que a informação manipulada mantenha

todas as características originais estabelecidas pelo proprietário da informação, incluindo controle de mudanças e garantia do seu ciclo de vida (nascimento, manutenção e destruição).

- **Disponibilidade**: propriedade que garante que a informação esteja sempre disponível para o uso legítimo, ou seja, por aqueles usuários autorizados pelo proprietário da informação.

- **Autenticidade**: propriedade que permite identificar que uma mensagem recebida foi originada da fonte anunciada e que não foi alvo de modificações ao longo do processo.

De modo geral, a segurança pode assumir um maior ou menor grau de importância de acordo com diversos fatores, dentre os quais pode-se destacar:

- **Ambiente físico**: dependendo do ambiente em que a aplicação se encontra, a segurança pode não ser prioridade. A segurança necessária quando se está em uma sala de aula ou conferência, por exemplo, será menor do que na praça de alimentação de um shopping. Isso porque possivelmente as pessoas do primeiro ambiente são mais confiáveis.

- **Requisitos de Quality of Service (QoS) da aplicação**: a aplicação, por ter conhecimento do tipo de informação com o qual está lidando, pode optar por estabelecer diferentes níveis de segurança ao longo do tempo e do tipo de dado processado. Uma aplicação que realiza troca de arquivos mp3 entre dois dispositivos não exige o mesmo nível de segurança que uma aplicação que realiza o pagamento da conta do restaurante utilizando o cartão de crédito.

- **Hardware**: computacionalmente, algoritmos de segurança (especialmente os de criptografia) exigem mais recursos da plataforma computacional. Assim, não é razoável esperar que um sensor responsável pela leitura da temperatura de uma casa utilize a mesma segurança que um *notebook*, onde aplicações mais robustas podem ser executadas.

Ao implantar uma solução de segurança, todos esses fatores devem ser levados em consideração de modo que seja alcançado um nível de segurança satisfatório. Além disso, em cenários como as redes sem fio, em que o ambiente e os dispositivos são muito diversificados, é interessante que seja adotada uma solução capaz se adaptar dinamicamente às mudança dos fatores anteriormente citados. Algoritmos ou sistemas capazes de responder a mudanças são o assunto da próxima seção.

2.2 Computação Ciente de Contexto

A computação ciente ou sensível ao contexto (*Context-Aware Computing*) define uma área de pesquisa relativamente recente, que possui aplicações em diferentes cenários computacionais e que apresenta desafios científicos importantes, os quais têm sido alvo da atenção de pesquisadores de diferentes áreas. A proposta é, em linhas gerais, elaborar uma maneira de coletar entradas capazes de refletir as condições atuais do usuário, do ambiente no qual se encontra e do próprio dispositivo computacional utilizado, considerando tanto suas características de hardware, como também de software e de comunicação Loureiro [2009]. Tais entradas são os chamados contextos. A partir do início da década de 90, foram publicados vários trabalhos que apresentaram definições para o termo contexto Silva [2010].

Dey [2001] formalizou a definição de contexto como sendo

> *"Qualquer informação que possa ser utilizada para caracterizar a situação de entidades (pessoa, lugar ou objeto) que sejam consideradas relevantes para interação entre um usuário e uma aplicação (incluindo o usuário e a aplicação)."*

Pode-se observar que a definição é bastante imparcial quanto aos tipos e variedades de dados que podem ser considerados como contextos, sendo ampla o suficiente para aceitar as diversas necessidades específicas de cada aplicação. Além disso, Dey também não restringe as fontes de contextos possíveis de serem utilizadas, permitindo que tais dados reflitam a situação de qualquer entidade relevante para cada caso em particular. Finalmente, é interessante notar que Dey apresenta uma definição precisa para o termo, sem listar tipos ou classes específicas de contextos.

Além das definições para a área, foram propostas também na literatura caracterizações para diferenciar os diversos tipos de contextos. Abaixo estão as definições de acordo com Indulska and Sutton [2003]:

- físicos: os dados são obtidos por meio de sensores ou algum outro tipo de hardware especializado;
- virtuais: prevêem dados a partir de aplicações e serviços de software;
- lógicos: são originados do processamento de alguma técnica ou algoritmo capaz de relacionar, inferir ou agrupar diversas fontes de dados para gerar informações contextuais.

No trabalho aqui apresentado, serão considerados contextos físicos (nível de bateria, condição da rede) e também contextos virtuais (requisitos de QoS do usuário, requisitos de segurança).

Para tratar das questões do contexto e também da segurança – já citadas nas seções anteriores –, a solução apresentada neste trabalho é centralizada na arquitetura de um *middleware*, conceito que será abordado na próxima seção.

2.3 Middleware

Middleware ou mediador, no campo da computação distribuída, é um programa de computador que faz a mediação entre um software e demais aplicações. É utilizado para mover ou transportar informações e dados entre programas de diferentes protocolos de comunicação, plataformas e dependências do sistema operacional. É geralmente constituído por módulos dotados com APIs de alto nível que proporcionam a sua integração com aplicações desenvolvidas em diversas linguagens de programação e interfaces de baixo nível que permitem a sua independência relativamente ao dispositivo. Seu objetivo é mascarar e heterogeneidade e fornecer um modelo de programação mais produtivo para os programadores de aplicativos. Além disso, o *middleware* é composto por um conjunto de processos ou objetos, que interagem entre si de forma a implementar comunicação e oferecer suporte para compartilhamento de recursos e aplicativos distribuídos.

O termo *middleware* pode também ser utilizado como designação genérica para referir-se aos sistemas de software que executam entre as aplicações e os sistemas operacionais. O objetivo do *middleware* é facilitar o desenvolvimento de aplicações, tipicamente as distribuídas, assim como facilitar a integração de sistemas legados ou desenvolvidos de forma não integrada.

No *middleware* proposto neste trabalho, uma das principais características é uma API que oferece de maneira transparente diferentes formas de comunicação às aplicações. Cada um dos tipos de comunicação possíveis serão detalhados na próxima seção.

2.4 Comunicação em Redes

Quando trata-se da comunicação entre dispositivos quaisquer que fazem parte de uma rede, podem ser considerados vários modelos computacionais. A seguir, serão apresentados os modelos de comunicação **Cliente-Servidor**, **Grupo** e **Anônima**.

Para um maior detalhamento sobre esses modelos, é indicada a leitura do capítulo

2 do livro *Distributed Operating Systems* Tanenbaum [1994].

2.4.1 Comunicação Cliente-Servidor

Nesse modelo, há uma clara distinção entre os nós que são provedores de um recurso ou serviço, chamados **servidores** e nós que requisitam esse serviço, chamados de **clientes**. Para evitar o grande *overhead* de protocolos orientados à conexão, o modelo cliente-servidor normalmente é baseado em um protocolo do tipo "requisita/responde". O cliente manda uma mensagem com a requisição desejada e o servidor responde com o que foi solicitado ou com uma mensagem de erro indicando algum problema na solicitação.

Muitos protocolos seguem esse modelo, entre eles os protocolos HTTP (utilizado pelos navegadores) e SMTP (para envio de e-mail).

Como principais vantagens desse modelo, destacam-se:

- simplicidade
- eficiência
- maior segurança, já que a maior parte dos dados fica armazenada no servidor
- atende a clientes com diferentes capacidades

Por outro lado, a simplicidade dessa abordagem trás algumas desvantagens, dentre as quais cita-se:

- possibilidade de sobrecarga do servidor
- pouca robustez, uma vez que uma falha no servidor compromete todo o funcionamento

2.4.2 Comunicação em Grupo

A abstração de Comunicação em Grupo tem por objetivo resolver problemas básicos de inconsistências na comunicação entre processos distribuídos que cooperam para a execução de uma tarefa. Nesse sentido, um grupo é apenas um conjunto de processos que cooperam. A principal propriedade de um grupo é que quando uma mensagem é

enviada para o grupo, todos os membros a recebem. Essa forma de comunicação acontece de um emissor para vários destinatários, diferente da comunicação ponto-a-ponto.

Como o envio de mensagens é implementado depende de alguns fatores, especialmente *hardware*. Mas, na maioria dos casos, é atribuído ao grupo um endereço especial, chamado endereço de *multicasting* – múltiplos destinos. Quando uma mensagem é enviada tendo como destino o endereço do grupo, todos os nós que pertencem ao grupo no momento recebem a mensagem.

Também é possível enviar a mensagem utilizando endereço de *broadcasting*. Nesse caso, qualquer dispositivo alcançável recebe a mensagem. Cabe então ao membro receptor saber se a mensagem era ou não destinada a ele e descartar a mensagem, caso necessário.

Para as duas implementações citadas, apenas uma mensagem é enviada, tendo como destino um endereço que representa ou o grupo ou todos os dispositivos alcançáveis. Outra alternativa é o envio de uma mensagem para cada membro do grupo, utilizando, nesse caso, um endereço de *unicast* – destino único. Essa abordagem é menos eficiente porque o número de pacotes enviados cresce com o número de nós da rede.

2.4.3 Comunicação Anônima

O termo **anônimo** vem do grego *anonymia* e significa "sem nome". Normalmente o anonimato está relacionado com a capacidade de uma entidade em manter sua identidade oculta dos demais componentes da rede.

Há vários motivos pelos quais uma entidade desejaria manter sua identidade oculta. Algumas delas são legais, legítimas e aprovadas socialmente, como por exemplo denúncias anônimas e ações de caridade.

Uma mensagem anônima pode ser interpretada como uma mensagem que não guarda informações sobre seus remetentes ou possíveis destinatários. O problema de saber se mensagens consecutivas pertencem ou não ao mesmo remetente nos leva ao problema conhecido como autenticação.

Apesar de ser um conceito diferente, um pseudônimo também garante, com alguma restrição, a identidade legítima da entidade que o utiliza. Exemplos de pseudônimos são número de matrícula, apelidos ou endereço IP. Uma vantagem dessa abordagem é que permite o estabelecimento de um canal de comunicação duradouro, já que é possível identificar o parceiro da comunicação, mesmo sem conhecer sua identidade legítima.

A Internet de hoje em dia é repleta de casos de comunicação anônima, já que grande parte dos comentários em fóruns, grupos e redes sociais é feita através do uso de pseudônimos. A *Wikipedia* é um dos mais bem sucedidos exemplos de usuários que cola-

boram de forma anônima, identificados apenas pelo IP ou pelo nome de usuário utilizado.

Na prática, um anonimato total na Internet é impossível, já que através do endereço IP é possível rastrear a origem de uma mensagem. Existem serviços, que, através de técnicas distribuídas conseguem garantir um maior grau de anonimato. O *Tor* é um dos mais utilizados com esse propósito.

Em uma análise feita por Herrmann et al. [2009], foi mostrado que o *Tor* é mais resistente inclusive que as tradicionais redes VPN. Em uma comparação entre essas duas alternativas e também com o *JAP – Java Anonymous Proxy*, foi demonstrado que se um possível intruso estiver em uma rede WLAN, será mais difícil fazer uma análise de dados – na tentativa de quebrar a criptografia da comunicação – no *Tor* do que nas outras duas soluções.

2.5 Trabalhos Relacionados

Quando trata-se de comunicação móvel, é desejável que mecanismos de segurança sejam pouco intrusivos, de forma que seja quase transparente para o usuário. Com relação ao contexto, espera-se que o mecanismo de segurança adotado seja capaz de responder de forma satisfatória a alterações no contexto.

O trabalho proposto por Al-Muhtadi et al. [2003] trata bem essas duas propostas para segurança em um ambiente de aplicações ubíquas cientes de contexto. O Cerberus – mecanismo de segurança implementado pelos autores – integra:

- identificação: faz a ligação entre uma entidade e uma identidade;
- autenticação: fornece a garantia que uma entidade possui permissão de acesso ao sistema/aplicação;
- sensibilidade ao contexto: capacidade de reagir a mudanças que ocorram no ambiente;
- raciocínio: capacidade de inferir resultados a partir de observações no contexto.

A arquitetura do Cerberus possui quatro componentes principais que mapeiam os itens mencionados acima: 1) serviço de segurança, 2) infra-estrutura de contexto, 3) base de conhecimento para armazenamento de políticas de segurança e 4) mecanismo de inferência que realiza raciocínios automatizados e reforça as políticas de segurança. Além disso, o controle de acesso utiliza o conceito de confiança para determinar permissões a determinados recursos. Dependendo do tipo de autenticação utilizada (reconhecimento

de voz e face, crachá eletrônico, dispositivos com fio, impressão digital, etc.) a entidade autenticada terá maior ou menor permissão de acesso.

Sabe-se que cada algoritmo de segurança fornece um diferente grau de segurança. Algoritmos de chave assimétrica, por exemplo, oferecem maior confidencialidade que algoritmos de chave simétrica. Da mesma forma, quanto maior o tamanho da chave utilizada, maior o grau de segurança oferecido, já que é mais difícil para um atacante descobrir essa chave. Em termos de recursos exigidos (como processamento e memória), algoritmos de criptografia possuem maior custo que algoritmos de *hash*. Park et al. [2000] definem um conjunto de métricas que podem ser utilizadas para classificar algoritmos. Entre as métricas citadas estão: tamanho de chave, processamento, memória, largura de banda, importância relativa para a aplicação. Não é definido nenhum método para classificação, há apenas o levantamento das métricas que foram julgadas relevantes.

Também utilizando o conceito de relevância de algoritmos, Taddeo and Ferrante [2000] discutem a seleção de algoritmos de criptografia que melhor se adequam a determinadas restrições de recursos de *hardware*. A relevância leva em consideração características do algoritmo e requisitos da aplicação. A escolha dos melhores algoritmos é modelada como um problema da mochila na sua versão 0-1: cada algoritmo possui uma utilidade u_i e um consumo de energia e_i; a seleção então se resume a encontrar os algoritmos que caibam nas restrições de consumo de energia e que maximizem o valor da relevância total. Uma desvantagem do método proposto é que o custo de seleção desses algoritmos é alto, já que o problema da mochila 0-1 é *NP-difícil*, apesar de poder ser resolvido com programação dinâmica em tempo pseudo-polinomial Cormen et al. [2001]. Além disso, existe uma restrição que é o fato de a classificação ser aplicada somente a algoritmos de criptografia, não sendo apresentada alternativa para classificação de outras classes de algoritmos de segurança.

Com um foco diferente, restrito apenas às condições da rede, Agarwal and Wang [2006] apresentam um modelo de gerenciamento de políticas de segurança adaptativo. O processo de tomada de decisão utiliza um modelo semi-*markoviano* Bravetti and Gorrieri [2000], que utiliza conceitos como conjunto de estados, conjunto de ações para cada estado, regras de decisão, políticas, épocas de decisão e funções de custo Puterman [1994]. No contexto do trabalho, o módulo de tomada de decisão relacionado com a escolha das políticas toma suas decisões em diferentes instâncias de tempo, chamadas de épocas de decisão, baseado no *feedback* obtido do módulo de monitoramento de contexto. Um estado é composto da política configurada e da taxa de erro de *bits*. As ações entre estados são definidas como a troca ou não da política utilizada no estado atual. A função de custo é também diretamente relacionada com a taxa de erro de *bits*. Os resultados obtidos mostram que o modelo consegue responder às mudanças no meio e utilizar algoritmos mais fortes nos momentos em que a rede apresenta melhor desempenho, e utilizando algoritmos mais fracos (com menor *overhead*) em momentos mais críticos. Um ponto

negativo que pode ser observado é a simplicidade das métricas do modelo, que não inclui, por exemplo, os recursos físicos dos dispositivos.

Raissi [2006] desenvolveu uma pesquisa em que utiliza um método de seleção de algoritmos criptográficos baseado em técnicas de Algoritmos Genéticos. O seletor, que executa no servidor da arquitetura proposta, recebe requisições de diversos clientes com requisitos de segurança, velocidade e QoS. O histórico utilizado na elaboração do algoritmo genético é armazenado em um servidor de banco de dados. Por um lado, existe uma semelhança com o presente trabalho no que diz respeito à proposta de escolher os melhores algoritmos de segurança em tempo de execução. Por outro lado, são estudados apenas algoritmos de criptografia e a arquitetura sugerida é adequada somente para redes com infra-estrutura bem definida e com dispositivos com bons recursos de *hardware* para trabalharem como servidores.

O último trabalho aqui citado é a nossa principal referência. Por essa razão, será dedicada uma subseção exclusiva para descrevê-lo melhor.

2.5.1 Middleware de segurança adaptativo para computação móvel

Esse trabalho foi desenvolvido como projeto de mestrado de Rocha [2007]. A idéia central do *middleware* é a capacidade de alterar dinamicamente os protocolos de segurança utilizados entre duas entidades computacionais., de acordo com variações no contexto.

Para escolha do melhor protocolo, são definidos parâmetros e métricas. Um parâmetro é definido como:

> "... valor que carrega informações sobre o contexto em que o middleware está inserido, e que é usado como auxílio no processo de escolha de protocolos de segurança a serem utilizados."

Exemplos de parâmetros são: latência, memória, uso de CPU, bateria, requisitos de segurança (confidencialidade, integridade e autenticação) e requisitos de QoS (latência máxima, *overhead* máximo).

Já a métrica é definida como:

> "... valor associado a um algoritmo de segurança que define uma propriedade do mesmo. Uma métrica de um protocolo de segurança representa a

agregação dos valores daquela métrica para cada algoritmo contido no protocolo."

Cada algoritmo é associada a seis métricas:

1. Confidencialidade
2. Integridade
3. Autenticação
4. Uso de memória
5. Tempo de processamento
6. *Overhead* de rede

Há também relação entre algumas métricas e parâmetros, podendo ser direta ou indireta. Por exemplo, o requisito de latência baixa indica indiretamente a necessidade de utilizar um algoritmo com baixo tempo de processamento. Já uma métrica de uso de memória está diretamente relacionada a um parâmetro de memória.

No momento da instalação do *middleware* em um dispositivo, são executados testes para calcular as métricas de cada algoritmo (memória, processamento e *overhead*). As demais métricas (de segurança) são definidas em um arquivo de configuração. Durante a inicialização do *middleware*, são gerados também todos os possíveis protocolos de segurança, além da leitura dos parâmetros fixos do dispositivos (capacidades de *hardware*).

Quando dois dispositivos desejam se comunicar, eles estabelecem uma conexão segura e trocam informações sobre seus parâmetros. Depois dessa troca, os dispositivos escolhem um subconjunto de protocolos que atendam as restrições dos parâmetros de ambos. Baseados então nos protocolos escolhidos, os dispositivos realizam a troca de chaves, que são necessárias para alguns tipos de algoritmos. Assim finaliza-se o processo de estabelecimento de conexão segura.

A partir daí, para cada nova transmissão, é escolhido um protocolo dentre aqueles já filtrados durante o processo de conexão. A mensagem é então enviada utilizando o protocolo escolhido.

Apesar dos ótimos resultados alcançados pelo *middleware* e pela sua capacidade de responder bem ao contexto, ele apresenta algumas restrições. Uma delas é o fato da implementação de referência ser limitada a apenas dois dispositivos, o que significa que não é possível utilizar o *middleware* para realizar comunicação em grupo. Outro problema é o fato da arquitetura proposta ser pouco modularizada, dificultando bastante a manutenção e expansão do *middleware*.

Capítulo 3

Modelos e Definições

Neste capítulo, serão apresentados modelos e definições considerados na implementação do presente trabalho.

3.1 Serviço de Segurança

A fim de tentar esclarecer melhor o termo *Protocolo de Segurança*, definido anteriormente por Rocha [2007], adota-se um novo termo: Serviço de Segurança.

Definição 1. *Um **Serviço de Segurança** é um conjunto não-vazio de **algoritmos** Rocha [2007] (Definição 3.1) que podem ser aplicados sobre uma mensagem enviada entre dois dispositivos quaisquer. Cada um dos algoritmos pode prover uma das três diretivas básicas de segurança: integridade, confidencialidade ou autenticidade.*

Assim, fica mantida a definição original de *Algoritmo de Segurança*, mas passa a ser adotado o termo *Serviço* em substituição ao termo *Protocolo*.

3.2 Modelo de Rede

Neste trabalho, são consideradas redes com algum tipo de infra-estrutura, como redes WLAN, ou redes de telefonia móvel (por exemplo, GSM). Com relação aos dispositivos, aplicações e protocolos suportados, são consideradas as seguintes possibilidades:

- **Dispositivos – hardware**: *Smartphones* com alguns MBytes de memória interna e processadores com pequena capacidade de processamento; Notebooks e/ou Netbooks com Gbytes de memória e uma maior capacidade de processamento;

- **Aplicações – software**: as aplicações podem ser divididas em dois subgrupos:
  1. Aplicações com envio de dados esporádico, como, por exemplo, o envio de mensagens SMS de forma segura;
  2. Aplicações de maior fluxo de dados, como transferência de arquivos ou *chats*.
- **Protocolos de comunicação**: IEEE 802.11 (WiFi) com ou sem segurança nativa, *Bluetooth* e GPRS

3.3 Modelo de Comunicação em Grupo

Como discutido na Seção 2.4.2, a comunicação em grupo tem como principal característica o fato de que uma mensagem, quando enviada, é destinada a todos os membros do grupo.

Para que seja feito o envio de uma única mensagem segura para todo o grupo (*multicast*), é necessário descartar a utilização de algoritmos de criptografia de chave assimétrica. Estes algoritmos, no geral, funcionam da seguinte forma:

1. Cada dispositivo gera um par de chaves: uma privada e uma pública
2. A chave pública é divulgada para todos os membros
3. A chave privada é mantida em segredo (apenas o dispositivo gerador a conhece)
4. Quando um dispositivo A deseja enviar uma mensagem para o dispositivo B, a mensagem é criptografada com a chave pública de B, de forma que apenas B será capaz de decriptografar essa mensagem utilizando sua chave privada

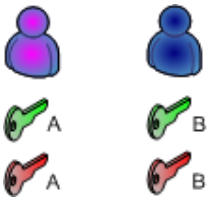
A Figura 3.1 ilustra o funcionamento descrito.

Ao aplicar algoritmos que utilizam esse tipo de chave, não é possível enviar uma única mensagem para todo o grupo (Figura 3.2b), é necessário enviar uma mensagem individual para cada membro utilizando a chave pública do destinatário (Figura 3.2a).

Outra decisão feita diz respeito a quem pode enviar mensagens para o grupo. Quando um grupo só aceita mensagens dos membros que o compõe, esse grupo é chamado de *fechado*. Quando aceita mensagens de dispositivos externos, é dito *aberto*. Na solução proposta, é considerado que o grupo é aberto.

Para grupos abertos, os dispositivos "não-membros" podem ou não ser anônimos. Em outras palavras, o grupo pode ou não exigir que um dispositivo externo apresente

(a) Cada dispositivo gera um par de chaves pública/privada.



(b) Os dispositivos trocam / divulgam suas chaves públicas.

(c) Dispositivo A envia uma mensagem para B criptografando com a chave pública de B. Apenas a chave privada de B é capaz de decodificar a mensagem.

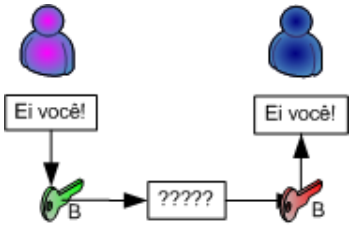
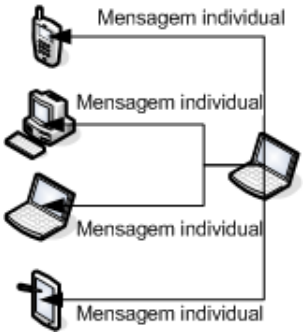


Figura 3.1: Processo de troca de chaves assimétricas. Chaves verdes são públicas, enquanto as vermelhas são privadas.

(a) Envio de mensagens em unicast.



(b) Envio de mensagens em multicast.

Figura 3.2: Formas de envio de mensagens em grupo.

alguma identificação para que sua mensagem seja aceita pelo grupo. Não será considerado o caso de um membro do grupo que comunique-se de forma anônima. Apesar de ser possível tratar essa situação com a solução proposta, avalia-se que o maior interesse de envio de mensagens anônimas exista em dispositivos "não-membros". O modelo de comunicação anônima será detalhado na seção 3.4.

Quanto à estrutura interna do grupo, pode-se classificá-lo em dois tipos:

1. ***Peer***: todos os dispositivos são iguais, não existe um líder e as decisões são feitas de forma coletiva;

2. **Hierárquico**: um ou mais dispositivos são responsáveis por tomar as decisões e repassá-las aos demais;

No *middleware*, existe um processo para tomada de decisões responsável, entre outras coisas, por escolher o melhor serviço de segurança para ser utilizado na comunicação do grupo a cada instante. Esse processo recebe como entrada os parâmetros de contexto de cada membro do grupo. Em um grupo sem nenhuma hierarquia, a tomada de decisão é mais complicada e acarreta em mais atraso e *overhead*, já que cada um dos membros deve ser consultado. Dessa forma, optou-se pela adoção de um grupo hierárquico, em que um nó tem um papel especial, de líder, capaz de tomar decisões em nome de todo o grupo.

Além de ser responsável por tomar decisões para o grupo, o líder também funciona como intermediador no processo de comunicação anônima, como será explicado na seção 3.4. Atualmente, o líder é o dispositivo que inicia a conexão em grupo, não existindo nenhum algoritmo específico para fazer um rodízio de dispositivos na liderança. Para o *middleware*, basta que exista um líder em determinado momento, não importando a forma como ele é escolhido ou gerenciado. Portanto, optou-se por essa abordagem simplificada de "eleição de líder".

Uma propriedade importante na comunicação em grupo é a capacidade da entrega atômica de mensagens. Isso significa que, ou todos os membros recebem uma mensagem, ou nenhum deles recebe. O modelo proposto não garante essa propriedade, podendo ocorrer casos em que um determinado membro não receba uma mensagem. A implementação de entrega atômica é um possível trabalho futuro para este trabalho.

Outra característica interessante diz respeito ao envio de mensagens ordenadas. Em alguns casos, é desejável a capacidade de determinar a ordem em que as mensagens serão entregues para cada dispositivo. Com o uso de algoritmos de criptografia, foi obtida uma implementação simples para garantir a ordem de recebimento das mensagens. Supõe-se que os membros do grupo conhecem a ordem na qual a mensagem deve ser entregue. Os detalhes do algoritmo para envio de mensagens ordenadas será tratado na seção 4.3.4.

Por fim, nossa modelagem de comunicação em grupo não considera o caso de grupos sobrepostos, que é o caso em que um membro participa ao mesmo tempo de dois ou mais grupos. Foi considerada aqui a existência de um único grupo e, nesse caso, ou o dispositivo pertence a um grupo, ou não pertence a nenhum.

3.4 Modelo de Comunicação Anônima

Existem diversas maneiras de garantir o anonimato de uma entidade na rede, como visto na seção 2.4.3. Será apresentado aqui um modelo de envio de mensagens anônimas baseado no uso de uma entidade centralizadora.

Hoje em dia, na Internet, é visto com frequência o uso de certificados digitais para

garantir a autenticidade de um *site*. Existem duas infra-estruturas principais:

- **Infraestrutura de Chaves Públicas (ICP)**: uma entidade central, chamada de Autoridade Certificadora, é responsável por assinar o certificado que foi emitido;
- **Teia de Confiança**: o certificado é assinado pela própria entidade e assinado também por outros que dizem confiar naquela entidade.

As assinaturas contidas em um certificado são garantias feitas por uma entidade que diz confiar nos dados contidos naquele certificado.

Neste modelo, é atribuído ao líder um papel semelhante ao da Autoridade Certificadora. Ele é quem decide se mensagens externas (recebidas de dispositivos anônimos) serão ou não encaminhadas ao grupo. Visto de outra forma, o líder garante sua confiança naqueles dados.

Quando um dispositivo deseja enviar mensagens ao grupo de forma anônima, o primeiro passo é estabelecer uma conexão segura com o líder do grupo. Através dessa conexão, o dispositivo externo ao grupo envia mensagens ao líder solicitando que sejam encaminhadas para o restante do grupo. Dessa forma, não só o dispositivo anônimo tem sua identidade protegida, mas o próprio grupo, já que, com exceção do próprio líder, não é passada nenhuma informação sobre os componentes do grupo.

No modelo atual, não existe um algoritmo ou mesmo método de aceitação de comunicação anônima por parte do líder. O único critério que o líder leva em consideração é a sua capacidade (em termos de recursos) de aceitar conexões externas ao grupo. Se o líder, por exemplo, encontra-se sobrecarregado, ele recusa esse tipo de comunicação. Futuramente, pode ser considerado o nível de maturidade (ou histórico) da relação entre o líder e o dispositivo solicitante. Além disso, pode ser delegada aos dispositivos a função de pontuar as mensagens recebidas, mesmo que eles não saibam de quem veio a mensagem originalmente. Agregando essas pontuações, o líder pode gerar uma espécie de avaliação geral sobre o dispositivo anônimo.

Capítulo 4

Solução Proposta

Considerando o *middleware* original, existe aqui um novo desafio: escolher o melhor serviço de segurança para um grupo de dispositivos, não apenas para um par de dispositivos.

Serão abordados neste capítulo os meios para configuração do *middleware*, a arquitetura da solução proposta e uma breve discussão sobre a plataforma **Android**. Serão mostrados também detalhes sobre a criação de uma nova estrutura de pacotes (semânticos) e sobre o processo de escolha do melhor serviço de segurança considerando todos os membros do grupo.

4.1 Configuração

Para armazenamento dos arquivos de configuração, optou-se por utilizar arquivos XML, já que essa linguagem de marcação possui características que, para este fim, tornam-na melhor que arquivos de texto simples:

- Separação do conteúdo da formatação
- Simplicidade e Legibilidade, tanto para humanos quanto para computadores
- Possibilidade de criação de *tags* sem limitação
- Criação de arquivos para validação de estrutura (DTD's)

Atualmente, existem três arquivos de configuração necessários para o funcionamento do *middleware*:

1. algorithms.xml
2. application.xml
3. policy.xml

Cada um desses arquivos possui uma finalidade específica, que será mostrada nas seções seguintes.

4.1.1 Arquivo algorithms.xml

Esse arquivo armazena informação sobre quais algoritmos podem ser utilizados pelos serviços de segurança. Cada entrada de um algoritmo armazena informações como:

- classe que o implementa;
- nome do algoritmo;
- tipo de chave utilizada;
- tamanho da chave necessária;
- dependências de outros algoritmos;
- recursos utilizados;
- uso do processador;
- *overhead* sobre a mensagem.

A Listagem 4.1 mostra um exemplo de entrada no arquivo XML representando um algoritmo de segurança. No caso, o algoritmo é o Data Encryption Standard (DES), com modo de operação *ECB*, implementado pela classe *DESECB*. Esse algoritmo possui uma chave simétrica – *SimmetricKey* – e, apesar de ser um algoritmo que tem como objetivo principal a garantia da confidencialidade, ele também garante – **offers** – algum grau de integridade. A "força" do algoritmo é determinada pela *tag* **strength**.

Apesar do arquivo inicial de configuração vir com valores pré-configurados, um provável utilizador deste *software* pode modificá-lo de acordo com suas necessidades, acrescentando ou removendo algoritmos ou modificando a "força" de determinados algoritmos, por exemplo.

```
1 <algorithm name="des_ecb" class="br.ufmg.dcc.rande.algorithms.crypto.DESECB">
2   <key class="br.ufmg.dcc.rande.algorithms.key.SymmetricKey" size="8" />
3   <resources memory="10000" negotbytes="0" negotsteps="0" />
4   <strength confidentiality="50" integrity="20" authenticity="0" />
5   <offers>
6     <dependency class="br.ufmg.dcc.rande.algorithms.extra.offer.IntegrityOffer" />
7   </offers>
8   <processing>
9     <process size="32" time="10.6" />
```

```
10      <process size="512" time="19.1" />
11      <process size="1024" time="35.1" />
12      <process size="32768" time="1043.5" />
13      <process size="131072" time="4180.9" />
14    </processing>
15    <network>
16      <overhead block="32" size="8" /> <!-- 32 B -->
17      <overhead block="512" size="8" /> <!-- 512 B -->
18      <overhead block="1024" size="8" /> <!-- 1 KB -->
19      <overhead block="32768" size="8" /> <!-- 32 KB -->
20      <overhead block="131072" size="8" /> <!-- 128 KB -->
21    </network>
22  </algorithm>
```

Listing 4.1: Exemplo de um algoritmo no arquivo XML.

4.1.2 Arquivo application.xml

Cada aplicação pode definir seus próprios requisitos de QoS e níveis de segurança desejados. A Listagem 4.2 mostra um exemplo de arquivo definido pela aplicação. Os níveis de segurança variam de 1 (não desejado) a 5 (obrigatório). Caso a aplicação não defina algum dos parâmetros, o *middleware* assumirá valores *default*.

```
1  <application>
2    <parameters>
3      <confidentiality value="3" /> <!-- from 1 to 5 -->
4      <integrity value="3" /> <!-- from 1 to 5 -->
5      <authenticity value="3" /> <!-- from 1 to 5 -->
6      <overhead value="512" /> <!-- 512 bytes -->
7      <handshake value="5000" /> <!-- 5000 bytes -->
8      <latency value="20000" /><!-- 20 ms -->
9      <buffer value="50000" /><!-- 50 KB -->
10   </parameters>
11 </application>
```

Listing 4.2: Exemplo de arquivo de configuração da aplicação.

4.1.3 Arquivo policy.xml

Esse arquivo determina os algoritmos (classes) que serão utilizados para escolher os serviços de segurança. Existem dois momentos de escolha:

1. **Estabelecimento de conexão segura entre dois ou mais dispositivos**: nesse momento, serviços que não atendem aos requisitos definidos pela aplicação (application.xml) ou que exigem mais recursos do que o total disponível no dispositivo são descartados. Essa é a política **estática** (já que só é executada em um único momento) de escolha dos serviços;

2. **Envio de mensagem**: para cada nova mensagem que precisa ser enviada, dados são coletados do contexto atual para avaliar qual o melhor serviço para envio da mensagem nesse momento. Como essa política é executada para cada envio, recebe a classificação de **dinâmica**.

Além disso, as políticas podem ser diferentes tanto para conexão par-a-par (*single*) quanto para conexões em grupo. Assim, tem-se um total de quatro políticas possíveis, todas mostradas no arquivo de exemplo da Listagem 4.3.

```
1  <policy>
2    <singleDynamicServiceChoice
3      class="br.ufmg.dcc.rande.network.security.policy.SingleDynamicServiceChoice" /
         >
4    <groupDynamicServiceChoice
5      class="br.ufmg.dcc.rande.network.security.policy.GroupDynamicServiceChoice" />
6    <singleStaticServiceChoice
7      class="br.ufmg.dcc.rande.network.security.policy.SingleStaticServiceChoice" />
8    <groupStaticServiceChoice
9      class="br.ufmg.dcc.rande.network.security.policy.GroupStaticServiceChoice" />
10 </policy>
```

Listing 4.3: Exemplo de arquivo de configuração de políticas para escolha dos serviços.

4.2 Arquitetura

A arquitetura inicial proposta para o *middleware* é bastante simples, dividida em três partes: conexão segura, máquina de segurança e controle de parâmetros. Apesar da abordagem simples facilitar o entendimento, ela peca no sentido de permitir novas extensões ou adaptações da implementação de referência. Por isso, é proposta aqui uma arquitetura mais flexível, composta de seis módulos:

1. **Algoritmos de Segurança**: engloba a implementação de todos os algoritmos de segurança.

2. **Analisador de Pacotes**: entidade responsável por receber pacotes e delegar a interpretação do pacote para o tratador correspondente;

3. **Conexão Segura**: inclui métodos para o estabelecimento da conexão segura (para-par ou em grupo) e também para a troca de chaves;

4. **Contexto**: contém informações sobre todos os parâmetros de contexto avaliados pelo *middleware*;

5. **Escolha de Serviços**: agrupa implementações relacionadas com tomadas de decisão, que atualmente referem-se à escolha dos melhores serviços de segurança para comunicação;

6. **Máquina de Segurança**: oferece serviços para a aplicação e gerencia as conexões seguras;

Com uma melhor separação de módulos, é possível isolar os componentes do *middleware* de forma que alterações em determinado ponto possam ser feitas sem causar grande impacto nos demais módulos.

A Figura 4.1 mostra um diagrama com o relacionamento dos módulos citados, juntamente com a *Aplicação* e o *Sistema Operacional*. Na figura, o *middleware* é representado pelo grande bloco central. Além dos módulos específicos do *middleware*, a arquitetura conta também com dois outros sistemas, representados na figura pelos blocos de *Semântica de dados* de Oliveira Costa [2008] e *Camada de acesso ao meio*.

Como pode-se perceber na figura, a aplicação acessa o *middleware* através de dois módulos: *Máquina de Segurança* e *Contexto*. A *Máquina de Segurança* é acessada para realizar diversas funções, como: estabelecimento de conexão segura e envio de mensagens seguras. Já o *Contexto* recebe da aplicação informações sobre requisitos de QoS e segurança. Além dos dados da *Aplicação*, o *Contexto* recebe dados de três outros módulos:

1. *Semântica de dados*: assim como a *Aplicação*, esse módulo informa requisitos de QoS e Segurança. A diferença está no fato de que os requisitos definidos nesse módulo podem ser especificados para cada bloco de mensagem, além de ser possível especificar requisitos que se repetem ao longo do tempo. Como exemplo, pode-se definir que os primeiros 2 KB de uma mensagem sejam enviados com nível de segurança máximo e que o restante da mensagem seja enviado com nível de segurança moderado. Essa abordagem é interessante quando as mensagens enviadas seguem algum padrão previamente estabelecido, como no caso de arquivos MPEG e MP3.

2. *Camada de acesso ao meio*: esse módulo informa a situação da rede, repassando por exemplo dados sobre interferência.

3. *Sistema Operacional*: é acessado pelo módulo de *Contexto* para coletar informações de *hardware* do dispositivo, como nível de bateria, memória disponível e capacidade do processador.

Voltando à *Máquina de Segurança*, percebe-se que ele acessa os módulos *Escolha de Serviços*, *Algoritmos de Segurança* e *Conexão Segura*. Quando a *Aplicação* deseja enviar uma mensagem, primeiramente ela realizar uma chamada à *Máquina de Segurança*. Esta, por sua vez, acessa o módulo *Escolha de Serviços*, que é responsável por escolher o melhor serviço de segurança para envio de uma dada mensagem. A escolha do melhor serviço leva em consideração os parâmetros armazenados no *Contexto*. Depois de escolhido o melhor serviço, a *Máquina de Segurança* faz uma chamada ao módulo de *Algoritmos de Segurança*, que aplica sobre a mensagem cada um dos algoritmos definidos pelo serviço. Da mesma forma, quando uma mensagem é recebida, a *Máquina de Segurança* acessa o *Algoritmos de Segurança* para aplicar os algortimos do serviço de modo reverso.

Com relação ao módulo de *Conexão Segura*, ele armazena informações sobre cada conexão estabelecida, além de oferecer funções para o estabelecimento de novas conexões e troca de chaves. O gerenciamento das conexões é feito pela *Máquina de Segurança*, que acessa o módulo de *Conexão Segura* a fim de consultar os dados de conexão ali armazenados.

Por fim, o módulo *Analisador de pacotes* é acionado sempre que uma nova mensagem é recebida. Ele fica responsável por interpretar o dado recebido e utilizar a *Máquina de Segurança* para realizar as ações necessárias.

Comparando com a arquitetura original do *middleware*, percebe-se que a idéia original foi mantida, uma vez que foram preservados os módulos *Máquina de Segurança*, *Conexão Segura* e *Controle de Parâmetros* (agora nomeado *Contexto*). O grande ganho da nova arquitetura está na melhor separação de funções, que aumenta a extensibilidade e também a facilidade de manutenção. Como cada módulo possui um papel bem definido, é também mais simples entender a comunicação entre eles. Além disso, a *Máquina de Segurança* deixou de ser um módulo extremamente sobrecarregado, repassando serviços a outros módulos e concentrando-se apenas no que deve ser oferecido à aplicação.

4.3 Implementação

4.3.1 Aplicação Móvel para Android

A fim de mostrar a viabilidade de uso do *middleware* por dispositivos móveis, foi implementada uma aplicação que pudesse ser executada em uma plataforma móvel. O Android foi escolhido como plataforma para desenvolvimento dessa aplicação considerando

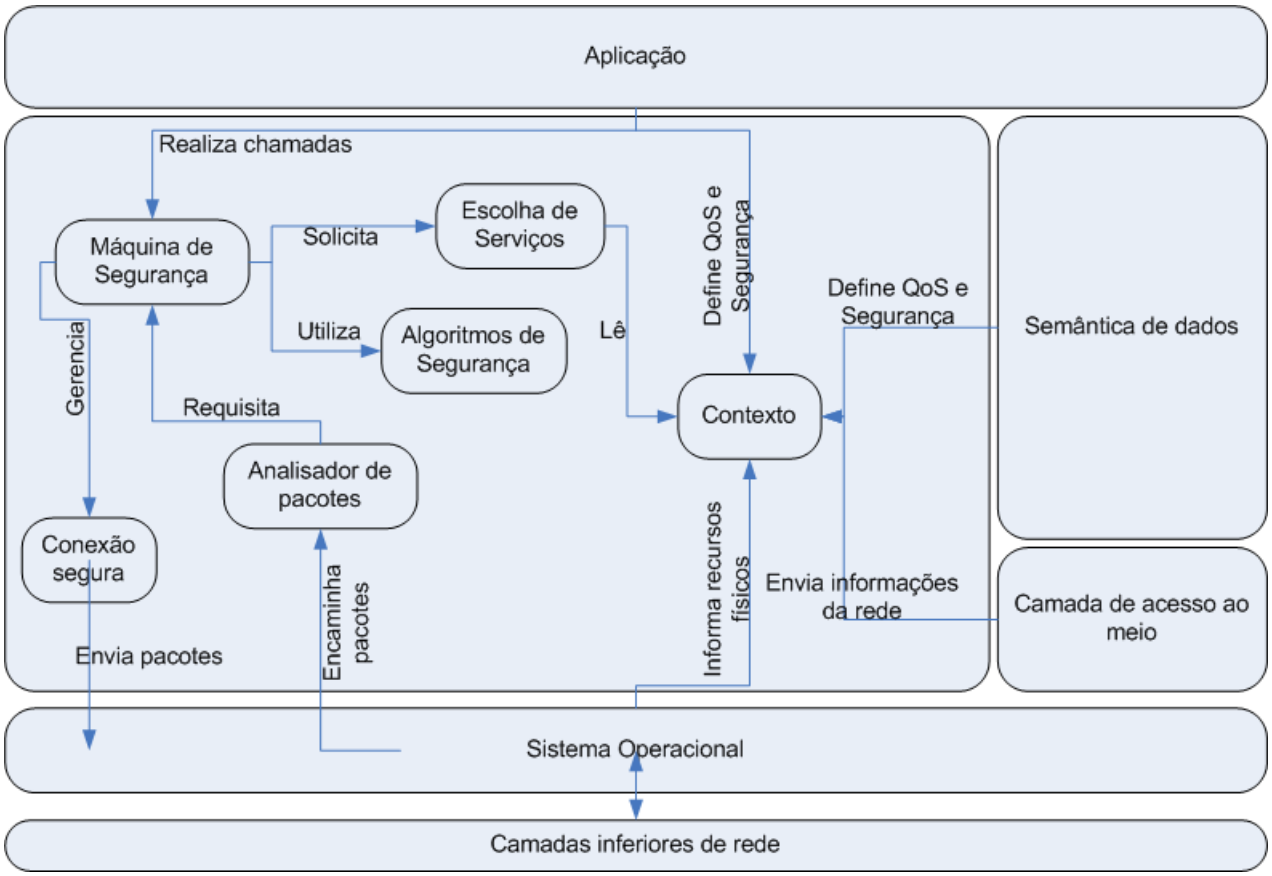


Figura 4.1: Arquitetura da solução proposta.

os seguintes critérios:

- Linguagem de programação disponível para o desenvolvimento das aplicações
- Facilidade de recuperar informações sobre o dispositivo
- Existência de emulador/simulador para teste das aplicações

O Android é um sistema operacional para dispositivos móveis mantido pela Google. Apesar de ser desenvolvido em várias linguagens de programação (C, C++ e Java), as aplicações desenvolvidas para essa plataforma são desenvolvidas em Java utilizando bibliotecas próprias da plataforma. Essas bibliotecas permitem acesso de forma fácil a informações sobre parâmetros do sistema, como quantidade de memória total, quantidade de memória livre, capacidade de processamento e nível de bateria disponível. A Google disponibiliza também um emulador para que sejam testadas as aplicações.

Como a linguagem de programação utilizada para desenvolvimento da aplicação é Java, foi decidido então migrar todo o *middleware* para essa linguagem, que tem com uma das principais características a portabilidade. Com a migração do *middleware* para essa linguagem, qualquer dispositivo que possua uma Java Virtual Machine (JVM) – abrangendo hoje vários *smartphones* e também *notebooks/netbooks*– pode usufruir dos

benefícios do *middleware*. O único módulo do que necessita de adaptação e que é específico de cada dispositivo é o módulo de *Contexto*, responsável por capturar parâmetros de sistema/*hardware*.

4.3.2 Benefícios da Nova Implementação

Ao portar a implementação de referência do *middleware* de C++ para Java, percebeu-se a oportunidade de realizar diversas melhorias. Nas subseções a seguir, serão detalhadas cada uma delas.

4.3.2.1 Programação orientada a interfaces

Uma interface garante que um contrato declarado deve ser seguido por todas as classes que a implementam. Assim, novas classes podem ser incorporadas sem afetar o funcionamento do *middleware*. Basta que para isso as interfaces sejam respeitadas. Algumas das interfaces criadas são:

- **SecurityAlgorithm**: qualquer algoritmo de segurança utilizado pelo *middleware*;
- **CryptoAlgorithm**: essa interface garante que toda classe que implemente um algoritmo de criptografia possua métodos para encriptar e decriptar mensagens;
- **AuthenticationAlgorithm**: essa interface garante que toda classe que implemente um algoritmo de autenticação possua métodos para assinar e verificar mensagens;
- **HashAlgorithm**: essa interface garante que toda classe que implemente um algoritmo de *hash* possua um método para gerar uma sequência de *bytes* a partir de uma dada mensagem;
- **StaticServiceChoice e DynamicServiceChoice**: interfaces relacionadas com a escolha dos melhores serviços de segurança a serem utilizados; essas interfaces garantem que as classes que as implementam possuam métodos para escolha dos melhores serviços de segurança. A configuração das classes relacionadas com essas interfaces foi mostrada anteriormente na Seção 4.1.

4.3.2.2 Uso de reflexão

Com a adoção dessa técnica, algoritmos e políticas de escolha de serviços podem ser adicionados ou mesmo substituir a implementação de referência sem a necessidade de alterar o código do *middleware*. Isso graças a uma característica da linguagem Java que é a reflexão. A partir das classes configuradas nos arquivos XML (Seções 4.1.1 e 4.1.3), é possível criar um novo objeto em tempo de execução. O correto funcionamento do *middleware* é garantido graças ao uso das interfaces, como descrito anteriormente.

4.3.2.3 Comunicação não-blocante

Um problema que existia no *middleware* era o fato dos dispositivos utilizarem uma forma de comunicação blocante. Isso significa que, ao enviar uma mensagem, o dispositivo entra em estado de inatividade (*CPU idle*) até receber uma resposta do destinatário. Uma desvantagem dessa abordagem é que um dispositivo pode não ser capaz de tratar novas conexões por já estar em processo de troca de mensagens com outro dispositivo.

Para contornar esse problema, duas soluções foram adotadas. A primeira delas diz respeito à implementação com o auxílio de *threads*. Para cada nova requisição que chega ao dispositivo (pedido de conexão, serviços de segurança, etc.), cria-se uma *thread* que fica responsável por tratar a informação recebida. Na arquitetura, essas *threads* pertencem ao módulo **Analisador de Pacotes**.

A segunda solução foi criar uma estrutura similar à estrutura de pacotes de dados, de forma que fosse possível identificar o tipo de informação que trafega na rede. De maneira geral, a estrutura desses pacotes contém:

- **Tipo do pacote**: um *byte* que identifica a classificação do pacote, de acordo com uma das seguintes possibilidades:
  1. Estabelecimento de conexão segura par-a-par;
  2. Estabelecimento de conexão segura em grupo;
  3. Troca de chaves par-a-par;
  4. Troca de chaves em grupo;
  5. Mensagem segura par-a-par;
  6. Mensagem segura em grupo;

7. Mensagem segura ordenada;
8. Mensagem anônima.

- **Subtipo do pacote**: para cada um dos tipos acima, existe um subtipo que distingue um pacote de outros da mesma classificação. Por exemplo, no estabelecimento de conexão segura, um pacote pode representar um pedido de nova conexão, o envio dos parâmetros de contexto do dispositivo, uma lista com os serviços de segurança que serão utilizados na conexão, etc.;
- **Dados específicos**: cada pacote possui informações que variam de acordo com seu tipo e subtipo. Por exemplo, um pacote de troca de chaves pode conter o tamanho da chave e a própria chave, enquanto um pacote de mensagem segura contêm o identificador do serviço de segurança usado e a própria mensagem.

Com a implementação dessas duas soluções, os dispositivos conseguem estabelecer conexões e trocas de chaves de forma paralela, já que os próprios pacotes contém as informações necessárias para a execução do processo.

4.3.3 Conexão

Durante o estabelecimento de uma conexão segura, é realizada uma filtragem dos serviços de segurança disponíveis. O objetivo é obter um número menor de serviços candidatos ao envio de mensagens, de forma que a escolha do melhor serviço para cada mensagem seja mais eficiente. Esse primeiro processo de decisão baseia-se nos parâmetros fixos dos dispositivos (recursos de *hardware*) e também de QoS.

Na primeira versão do *middleware*, o algoritmo de escolha proposto realiza uma espécie de "fusão" dos parâmetros físicos e requisitos de QoS, considerando sempre o valor mais restritivo. Os parâmetros analisados são:

- Parâmetros físicos:
  - Carga média do processador
  - Memória Disponível
- Requisitos de QoS:
  - Máximo de *buffer* utilizado
  - *Overhead* da mensagem

- *Overhead* de negociação das chaves do serviço
- Latência máxima
- Nível de confidencialidade
- Nível de integridade
- Nível de autenticidade

Quando a comunicação envolve apenas dois dispositivos, essa abordagem é eficiente e o *middleware* encontra um serviço para a maior parte da combinação de requisitos. Já no ambiente de grupos, pode-se ter dispositivos muito heterogêneos, o que significa uma maior diversidade de combinações de requisitos de QoS e parâmetros físicos. Podem acontecer situações em que, pelo fato de apenas um membro do grupo ser muito restritivo, os parâmetros finais combinados sejam também restritivos. Como resultado, o *middleware* poderia não ser capaz de encontrar um serviço que atendesse aos parâmetros avaliados.

Para evitar situações como essas, é proposta aqui uma nova forma de agregação dos parâmetros, baseada na média aritmética dos valores de cada um dos dispositivos. Não se espera que essa abordagem resolva o problema para todas as situações, mas acredita-se que seja mais eficiente para o ambiente de grupos.

Considerando um grupo composto por n dispositivos, e utilizando os parâmetros já definidos para o *middleware* por Rocha [2007] (listadas anteriormente), então o valor agregado VA_i para cada parâmetro p_i é definido por:

$$VA_i = \frac{\sum_{i=1}^{n} p_i}{n}$$

No capítulo 5, será apresentada uma comparação entre a abordagem de "fusão" e a de valor agregado baseado na média aritmética.

4.3.4 Transmissão

O processo de transmissão de pacotes anônimos, em grupo e ordenada difere da transmissão simples de dispositivos conectados par-a-par. A seguir são apresentados três algoritmos simples, em alto nível, mostrando como ocorre a transmissão em cada um desses casos.

No algoritmo 4.1, percebe-se que apenas o líder é capaz de escolher o melhor serviço de segurança para cada mensagem. Isso ocorre porque apenas o líder possui informações sobre os parâmetros e requisitos de QoS de todos os membros do grupo, que são necessários para a escolha do melhor serviço. Apesar de exigir o envio de uma mensagem de requisição

Algorithm 4.1 Algoritmo de transmissão de mensagens em grupo

```
1: if é líder do grupo then
2:     escolhe melhor serviço de segurança
3: else
4:     envia mensagem ao líder do grupo solicitando o melhor serviço de segurança
5:     recebe o identificador do serviço de segurança
6: aplica serviço de segurança na mensagem
7: envia a mensagem para todo o grupo
```

ao líder (1 *byte* de *overhead*) e uma mensagem de resposta indicando o melhor serviço, (5 *bytes* de *overhead*), evita-se que essas informações sejam armazenadas em todos os dispositivos. Dependendo do cenário, talvez seja mais interessante que essas informações sejam replicadas em todos os dispositivos, para diminuir o *overhead* da rede. No entanto, considera-se o *overhead* baixo para o cenário testado, com poucos dispositivos.

Algorithm 4.2 Algoritmo de transmissão de mensagens anônimas

```
1: if é anônimo then
2:     if não possui conexão segura par-a-par com o líder then
3:         estabelece conexão segura com o líder
4:     escolhe melhor serviço de segurança
5:     aplica serviço de segurança na mensagem
6:     envia a mensagem ao líder do grupo
7: else if não é anônimo then
8:     if é líder then
9:         recebe mensagem anônima
10:        decodifica mensagem
11:        escolhe melhor serviço para enviar a mensagem ao grupo
12:        aplica o serviço de segurança na mensagem
13:        envia a mensagem para o grupo
14:    else if mensagem pertence ao grupo then
15:        recebe mensagem
16:        decodifica a mensagem
```

No algoritmo 4.2, observa-se que, como já definido na Seção 3.4, um dispositivo só é capaz de enviar mensagens anônimas se possuir uma conexão segura par-a-par com o líder do grupo. Essa necessidade vem do fato de que os membros do grupo só recebem/identificam mensagens que sejam originadas de algum membro do grupo. Assim, o líder funciona como uma ponte entre o dispositivo anônimo e o restante do grupo.

Percebe-se no algoritmo 4.3 que, apesar de todos os nós receberem a mensagem – através do envio *multicast*, nenhum deles é capaz de interpretá-la, já que as chaves são mantidas em segredo pelo emissor. O que fica realmente garantido é que os membros só interpretarão a mensagem de acordo com uma ordem já conhecida pelo grupo. Os passos descritos nas linhas 15 e 16 são repetidos até que todos os membros tenham recebido a mensagem.

Algorithm 4.3 Algoritmo de transmissão ordenada

```
1: if envio de mensagem then
2:     if é líder do grupo then
3:         escolhe melhor serviço de segurança
4:     else
5:         envia mensagem ao líder do grupo solicitando o melhor serviço de segurança
6:         recebe o identificador do serviço de segurança
7:     gera as chaves necessárias para o serviço escolhido
8:     envia a mensagem em multicast para o grupo
9:     envia as chaves para o primeiro receptor de acordo com a ordem desejada
10:
11: else if recebimento de mensagem then
12:     if mensagem de dados then
13:         armazena a mensagem recebida
14:     else if mensagem de chaves then
15:         decodifica mensagem previamente armazenada
16:         envia as chaves para o próximo receptor de acordo com a ordem desejada
```

Há nesse processo um custo de processamento para cálculo das novas chaves em cada novo envio e também o *overhead* de mensagens para envio das chaves (uma mensagem para cada membro). No entanto, esse método é simples e foi incorporado diretamente ao *middleware*, o que significa que o serviço de segurança utilizado é definido sempre de forma dinâmica.

Capítulo 5

Resultados

Neste capítulo, são apresentados os principais resultados deste trabalho: o desenvolvimento de uma aplicação na plataforma Android demonstrando o funcionamento do *middleware* e sua real implantação em uma plataforma móvel; comparação de dois métodos diferentes de escolha dos serviços de segurança para comunicação em grupo.

5.1 Aplicação *Mensagem Segura*

A fim de validar a utilização do *middleware* em plataformas móveis, foi realizada a implementação de uma aplicação na plataforma Android. Além de validar a utilização do *middleware*, foi possível realizar a integração da leitura de dados de contexto disponibilizados pela plataforma. São disponibilizados dados como nível de bateria, quantidade de memória total, quantidade de memória disponível e carga do processador. Em suma, todos os parâmetros necessários ao *middleware*.

A aplicação foi implementada como uma *Activity* Android [2010], com três modos de operação: comunicação par-a-par, comunicação em grupo e comunicação anônima. Todos esses modos de operação acessam métodos fornecidos pela Application Programming Interface (API) do *middleware* para realização de cada um dos tipos de comunicação.

Para tratar o recebimento das mensagens, foi implementado um *Service* Android [2010], que tem como única finalidade encaminhar para o *middleware* o tratamento de todos os dados recebidos. As modificações de dados de contexto do dispositivo, como nível de bateria, são recebidas através da declaração de *Broadcast Receivers*. Todos estes conceitos fazem parte da arquitetura da plataforma Android; maiores detalhes podem ser obtidos na documentação oficial do projeto.

Os arquivos de configuração necessários (Seção 4.1) são armazenados no diretório raiz do disco rígido do aparelho e são lidos durante o processo de inicialização da aplicação. Futuramente, pode-se implementar uma funcionalidade para que o usuário consiga definir parâmetros de QoS e nível de confiança nos algoritmos através da própria aplicação, sem

a necessidade de edição manual desses arquivos. A forma de armazenamento pode ser também melhorada, utilizando o conceito de *Shared Preference* já existente na plataforma.

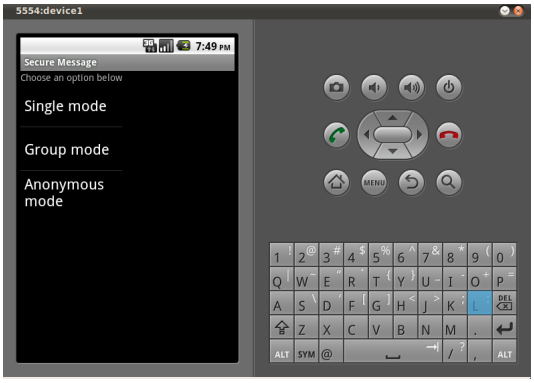
As funcionalidades oferecidas para o usuário variam de acordo com o modo de operação escolhido:

- **Comunicação par-a-par**
  - **Conectar**: inicia uma conexão par-a-par com dispositivos vizinhos;
  - **Conectar utilizando semântica de dados**: realiza conexão par-a-par utilizando a integração com o módulo de semântica de dados;
  - **Enviar mensagem**: envia uma mensagem para um destinatário específico.
- **Comunicação em grupo**
  - **Conectar**: inicia conexão em grupo com dispositivos vizinhos; o dispositivo que dispara a formação do grupo é definido como líder do grupo;
  - **Conectar utilizando semântica de dados**: realiza a conexão em grupo utilizando a integração com o módulo de semântica de dados;
  - **Enviar mensagem**: envia mensagem *multicast* para todo o grupo;
  - **Enviar mensagem ordenada**: envia mensagem garantindo ordenação de recebimento.
- **Comunicação anônima**
  - **Conectar-se ao líder do grupo**: inicia uma conexão par-a-par com o líder do grupo (se houver um grupo formado);
  - **Enviar mensagem anônima**: envia uma mensagem pro grupo garantindo anonimato do emissor.

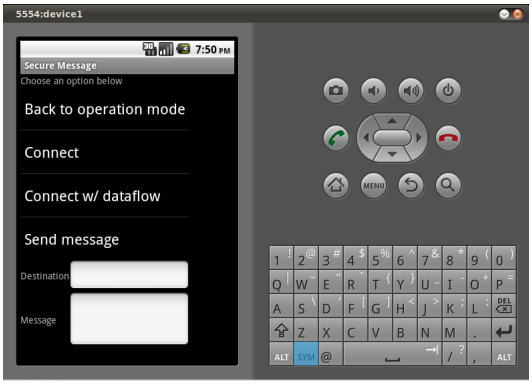
A seguir são apresentadas algumas figuras que ilustram a aplicação executando no simulador da plataforma. Este simulador, apesar de possuir algumas limitações, é utilizado para realização de testes em um ambiente similar a um dispositivo real.

Como pode-se observar na Figura 5.2, novas mensagens são notificadas ao usuário através da barra de *status* do sistema. Uma possível melhoria é armazenar todas as mensagens em memória para que o usuário possa, em um momento posterior, consultá-las. Além das mensagens, alertas são emitidos quando há estabelecimento de novas conexões ou problemas como impossibilidade de encontrar um serviço que atenda às restrições estabelecidas.

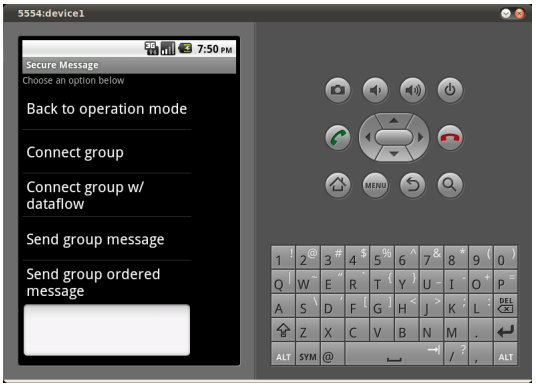
(a) Tela inicial da aplicação com os modos de operação disponíveis.



(b) Tela com as funcionalidades disponíveis no modo par-a-par (*Single Mode*).



(c) Tela com as funcionalidades disponíveis no modo em grupo.



(d) Tela com as funcionalidades disponíveis no modo anônimo.

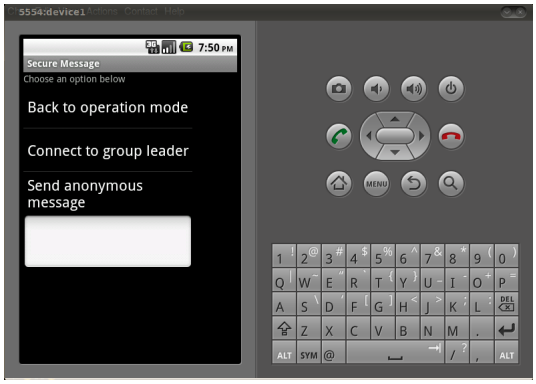


Figura 5.1: Modos de operação da aplicação.

5.2 Métodos de Escolha de Serviços de Segurança

Na Seção 4.3.3, foi proposta uma nova forma de agregação dos parâmetros de QoS e recursos dos dispositivos que compõem um grupo. Ao invés de utilizar sempre os parâmetros mais restritivos, foi desenvolvido um método de escolha baseado na média aritmética dos parâmetros de cada membro do grupo.

Para ilustrar o benefício dessa abordagem, considere o cenário ilustrado pela Tabela 5.1. Nela estão mostrados os valores dos parâmetros de três dispositivos, o valor mais restritivo (em vermelho) e a média aritmética. Os valores de confidencialidade, integridade e autenticidade variam de 1 a 5, sendo 5 o valor máximo de nível de segurança. Os parâmetros de *buffer* e *overhead*, medidos em *bytes*, indicam o valor máximo que o serviço de segurança pode utilizar. O valor de latência (ms) indica que o serviço de segurança não pode demorar mais que esse tempo para codificar a mensagem.

Utilizando os parâmetros de QoS mostrados na tabela, considerando dispositivos com capacidade de *hardware* idêntica e sujeitos às mesmas condições de rede (fixas para esse cenário), tenta-se enviar a seguinte mensagem: *"E agora? Estou tentando enviar*

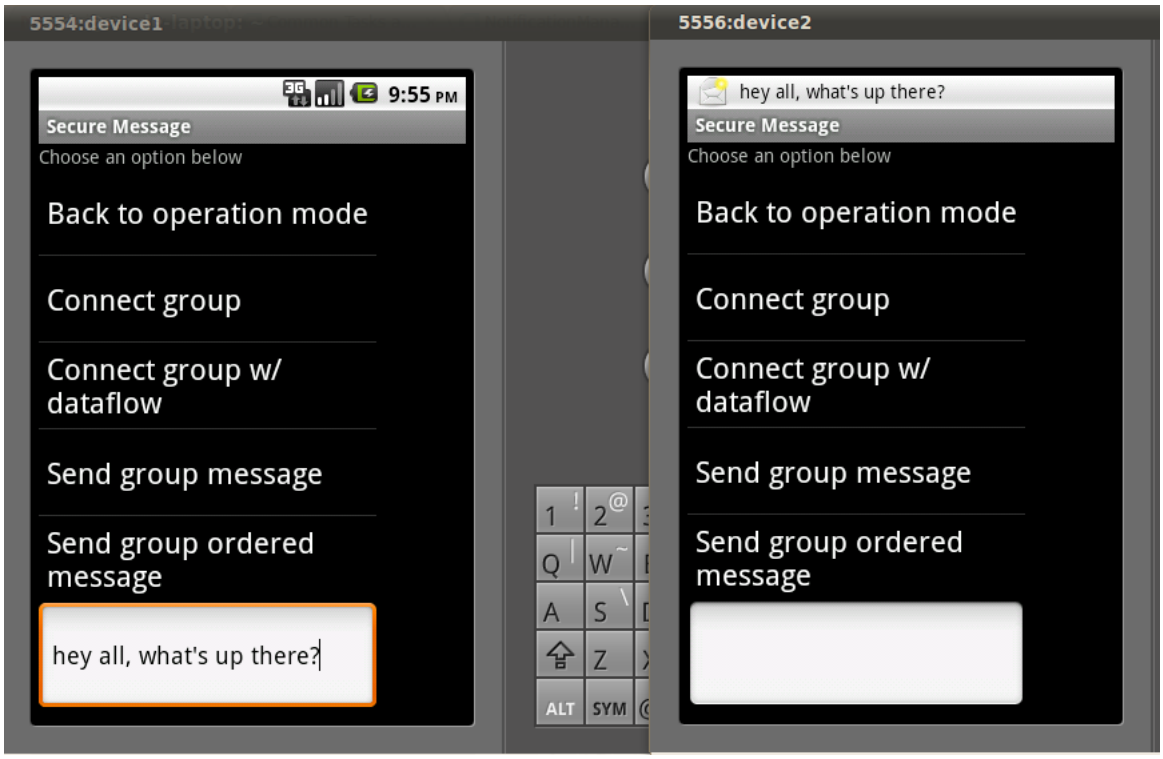


Figura 5.2: Exemplo de envio de mensagem em grupo. Destaque para a janela do dispositivo *device2*: no alto há um alerta indicando o recebimento da mensagem.

	Dispositivo 1	Dispositivo 2	Dispositivo 3	Média
Confidencialidade(1 a 5)	1	1	5	2
Integridade(1 a 5)	2	3	5	3
Autenticidade(1 a 5)	1	1	5	2
Máximo de overhead(bytes)	256	512	1024	597
Máximo de negociação(bytes)	256	512	1024	597
Máximo de *buffer*(bytes)	20000	50000	100000	53333
Latência máxima (ms)	2000	3000	4000	3000

Tabela 5.1: Parâmetros de QoS de três dispositivos, valor mais restritivo(em vermelho) e a média aritmética para cada atributo.

uma mensagem longa para testar uma nova estratégia diferente da abordagem inicialmente proposta, será que é possível? Vamos verificar em breve."

Com o método do valor mais restritivo, o *middleware* foi incapaz de encontrar um serviço de segurança que pudesse ser aplicado à mensagem. Ao adotar a estratégia de valores médios, acontece certo relaxamento dos parâmetros de QoS exigidos, como pode-se perceber comparando as colunas *Mais Restritivo* e *Média* da Tabela 5.1. Assim, ao tentar enviar a mesma mensagem, o *middleware* encontra, como melhor solução o serviço composto dos seguintes algoritmos: DES (modo ECB) com chave de 8 bytes para criptografia, SHA-1 para integridade e Digital Signature Algorithm (DSA) com chave de 128 bytes (1024 bits) para autenticidade.

Nota-se que esse serviço não atende em plenitude os requisitos individuais de cada dispositivo. No entanto, ao assumir a visão de coletividade do grupo, as exigências foram

relaxadas e encontrou-se uma forma de solucionar o conflito que existiria (impossibilidade de envio por não existir serviço adequado) caso fosse adotada a primeira estratégia.

A viabilidade de uso desta segunda estratégia pode variar de acordo com cada cenário. Em alguns casos, pode-se preferir não enviar uma mensagem a enviá-la utilizando um serviço mais fraco do que o desejado. Por outro lado, em ambientes muito heterogêneos, com grande número de dispositivos e variedade de *hardware*, pode ser interessante adotar uma estratégia com certo relaxamento.

Outra abordagem possível é realizar uma média ponderada dos requisitos de cada dispositivo. Os pesos utilizados poderiam ser baseados, entre outras coisas, no número de vezes em que o membro foi líder do grupo ou de acordo com a reputação do membro, como em redes P2P Wu and Wu [2009].

Capítulo 6

Considerações Finais

6.1 Conclusões

Neste trabalho, foi apresentada uma solução capaz de estabelecer conexões seguras entre membros de um grupo. Os serviços de segurança utilizados no envio de cada mensagem podem variar a todo momento, de acordo com os parâmetros definidos pelo contexto em que o usuário/aplicação se encontra.

Foi discutida também uma nova funcionalidade do *middleware* que é a possibilidade de envio de mensagens anônimas entre os dispositivos. Essas mensagens seguem a mesma arquitetura base do *middleware*, aproveitando assim do benefício de escolha dos serviços de segurança em tempo real. Outra funcionalidade mostrada na nova implementação foi a possibilidade de envio de mensagens ordenadas. Para garantir o recebimento (leitura) ordenado das mensagens, foi visto que um conjunto de chaves é gerado para cada novo envio.

Debateu-se também diferentes modos de escolha do melhor serviço quando trata-se da conexão em grupo. Através de casos de teste, ficou evidente que cenários específicos podem exigir diferentes estratégias. Com a nova estrutura de configuração de arquivos, é possível informar ao *middleware* qual abordagem deve ser utilizada.

Além disso, foi mostrada uma aplicação evidenciando a possibilidade de portar e utilizar o *middleware* proposto em dispositivos móveis. Para isso, foi utilizada a plataforma *Android*, que exigiu a migração do código original do *middleware* de C++ para Java. Melhorias foram também realizadas com o intuito de se obter maior extensibilidade e dinamicidade. Algoritmos podem agora ser integrados ao sistema sem que haja a necessidade de alterações no módulo da *Máquina de Segurança*.

6.2 Trabalhos Futuros

A implementação de referência apresentada aqui, tanto para o *middleware* quanto para a aplicação, possui vários pontos que podem ser melhorados.

Um desenvolvimento interdisciplinar que pode ser feito diz respeito à análise da interface da aplicação. Utilizando técnicas da área de Interação Humano-Computador (IHC), pode-se repensar a maneira como as informações são hoje apresentadas, de forma que tenha uma boa usabilidade e atenda às reais necessidades de usuários de dispositivos móveis.

Ainda sobre a aplicação, a plataforma *Android* possui recursos que abrem a possibilidade de existir um *menu de configuração*. Essa funcionalidade integraria todas as opções de configuração do *middleware*, como:

- Definição dos requisitos de QoS;
- Definição da política de escolha de serviços de segurança;
- Anotações do módulo de semântica de dados;
- Algoritmos de segurança utilizados

Com relação ao modelo de comunicação em grupo, podem-se rever duas características: atomicidade de mensagens e possibilidade de co-existência de diversos grupos (grupos sobrepostos). Atualmente, nenhuma destas é implementada pelo *middleware*. Outro ponto passível de melhorias diz respeito ao processo de escolha do líder do grupo. Atualmente, a liderança fica sob responsabilidade do membro que inicia o processo de formação do grupo. Seria interessante comparar diferentes algoritmos de eleição de líder, discutir parâmetros para a eleição e definir qual seria a melhor abordagem para o contexto de comunicação em grupo aqui apresentado.

Com relação ao *middleware*, vale a pena investir em uma melhor análise dos parâmetros utilizados. Uma nova métrica que poderia ser levada em consideração é o tamanho da chave. Quanto maior a chave utilizada por um algoritmo, mais difícil é para um atacante realizar análise de dados e descobrir a chave. Algoritmos que utilizam chaves da ordem de centenas de bits, como Rivest Shamir Adleman (RSA), DSA – chaves de 1024 e 2048) – são, em geral, mais seguros que algoritmos que usam chaves da ordem de alguns poucos *bytes*.

Referências

A.K. Agarwal and Wenye Wang. Dspm: Dynamic security policy management for optimizing performance in wireless networks. In *Military Communications Conference, 2006. MILCOM 2006. IEEE*, pages 1 –7, 2006.

Jalal Al-Muhtadi, Anand Ranganathan, Roy Campbell, and M. Dennis Mickunas. Cerberus: A context-aware security scheme for smart spaces, 2003.

Android. http://developer.android.com/guide/index.html, 2010.

Mario Bravetti and Roberto Gorrieri. The theory of interactive generalized semi-markov processes. In *Theoretical Computer Science*, pages 5–32, 2000.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2a edition, 2001.

Daniel Nogueira de Oliveira Costa. Adaptação de mecanismos de segurança para comunicação em ambientes móveis. Master's thesis, Universidade Federal de Minas Gerais, 2008.

A. K. Dey. Using and understanding context, 2001.

Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: Attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the ACM Cloud Computing Security Workshop (CCSW)*, New York, 2009.

J. Indulska and P. Sutton. Location management in pervasive systems, 2003.

A. A. F. et. al. Loureiro. Minicurso: Computação ubíqua ciente de contexto: Desafios e tendências, 2009.

Jong-Ho Park, Moon-Young Jung, and Seung-Woo Seo. Adaptive security support in future internet through class of security level, 2000.

M.L. Puterman. Markov decision processes: Discrete stochastic dynamic programming, 1994.

J. Raissi. Dynamic selection of optimal cryptographic algorithms in a runtime environment. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 184 –191, 2006.

Bruno Pontes Soares Rocha. Middleware de segurança adaptativo para computação móvel. Master's thesis, Universidade Federal de Minas Gerais, 2007.

Moray Rumney. Imt-advanced: 4g wireless takes shape in an olympic year. Technical report, Agilent Technologies, 2008.

T. R. M. B. Silva. Tratamento de conflitos coletivos em sistemas ubíquos cientes de contexto, 2010.

Antonio Vincenzo Taddeo and Alberto Ferrante. Run-time selection of security algorithms for networked devices, 2000.

Andrew Tanenbaum. *Distributed Operating Systems*. Prentice Hall, 3a edition, 1994.

Peng Wu and Guoxin Wu. Reputation mechanism in peer-to-peer network. In *Information Science and Engineering (ICISE), 2009 1st International Conference on*, pages 1793 – 1796, 2009.