UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Abimael Aldevino Fidêncio
André Ricardo Natal Simões

Votação Digital Segura em Dispositivos Móveis

Trabalho de Conclusão de Curso submetido à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Bacharel em Ciência da Computação.

Marcelo Luiz Brocardo
Orientador

Prof. Ricardo Felipe Custódio, Dr.
Co-Orientador

Florianópolis, junho de 2011

Votação Digital Segura em Dispositivos Móveis

Abimael Aldevino Fidêncio
André Ricardo Natal Simões

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção do título de Bacharel em Ciência da Computação e aprovada em sua forma final pelo Departamento de Informática e Estatística da Universidade Federal de Santa Catarina.

Prof. Vitório Bruno Mazzola, Dr.
Coordenador do Curso

Banca Examinadora

Marcelo Luiz Brocardo

Prof. Ricardo Felipe Custódio, Dr.

Vitório Bruno Mazzola, Dr.

Nossas dúvidas são traidoras e nos fazem perder o que, com freqüência, poderíamos ganhar, por simples medo de arriscar.
William Shakespeare

Oferecemos este trabalho aos nossos pais, por terem nos fornecido condições de estudar e chegarmos aonde chegamos.

Agradecimentos

Agradecemos ao nosso orientador Marcelo Luiz Brocardo pela confiança e assistência que nos forneceu durante o desenvolvimento do trabalho.

Ao professor Ricardo Felipe Custódio por viabilizar o projeto e por nos ter apresentado ao professor Roberto Samarone dos Santos Araújo.

Ao professor Roberto Samarone dos Santos Araújo pelo apoio acadêmico nos primeiros e fundamentais passos para este projeto.

Nosso agradecimento também ao Fabiano Castro Pereira por todo suporte com a rede de misturadores.

E, finalmente, um agradecimento especial aos nossos pais por acreditarem que a melhor herança que podem nos deixar é a educação. A eles, aos nossos irmãos e pessoas queridas dedicamos todo o nosso esforço em desenvolver este trabalho. Muito obrigado a todos!

Sumário

Lista de Figuras

Lista de Siglas

LabSEC	Laboratório de Segurança em Computação - UFSC
PIN	Número de Identificação Pessoal (Personal Identification Number)
ICP	Infraestrutura de Chaves Públicas (PKI - Public Key Infrastructure)
RNP	Rede Nacional de Ensino e Pesquisa
RSA	Rivest-Shamir-Adleman
SSL	Protocolo de Camada de Sockets Segura (Secure Sockets Layer)
TLS	Segurança da Camada de Transporte (Transport Layer Security)
PCV	Processo Cliente de Votação
PSV	Processo Servidor de Votação
JCE	Java Cryptographic Extension
ME	Plataforma para Dispositivos Móveis (Mobile Edition)
SE	Plataforma padrão (Standart Edition)
EE	Plataforma Empresarial (Enterprise Edition)
CLDC	Configuração para Dispositivos Móveis Limitados (Connected Limited Device Configuration)
CDC	Configuração para Dispositivos Móveis com maior poder computação (Connected Devide Configuration)
MIDP	Perfil JavaME para Dispositivos CLDC (Mobile Information Device Profile)
IMP	Subconjunto do MIDP utilizado em dispositivos com uma interface mais limitada (Information Module Profile)
KVM	Máquina Virtual Java utilizada em Dispositivos Móveis (K Virtual Machine)
IDE	Ambiente Integrado de Desenvolvimento (Integrated Development Environment)
PDA	Assistente Pessoal Digital (Personal digital assistants)
API	Interface de Programação de Aplicações (Application Programming Interface)
SATSA	API para Java ME com recursos de criptografia (The Security and Trust Services API)

Resumo

A necessidade da informação a toda hora e lugar, fez com que os dispositivos móveis, principalmente os celulares, se tornassem muito populares. Tendo em vista esse fato, cada vez mais, há necessidade de se desenvolverem variados tipos de aplicativos para os celulares.

O projeto tem por objetivo elaborar um protótipo de um sistema de votação via internet para celulares, tomando como estudo algoritmos de criptografia, visando a segurança do sistema como um todo, assim como respeitando a privacidade do usuário.

Palavras chave: Votação, Votação Segura, Dispositivos Móveis, Criptografia Assimétrica.

Abstract

The need for information all the time and place, made the devices, especially cell phones, became very popular. Considering this fact, increasingly, there is a need to develop various types of applications for mobile phones.

The project aims to develop a prototype of a voting system for internet mobile phones, using as a study of encryption algorithms, for the security of the system as a whole as well as respecting the privacy of the User.

Keywords: Vote, secure vote, mobile devices, asymmetric cryptography.

Capítulo 1

Introdução

A primeira vista, a mobilidade aparentava ser apenas mais uma facilidade, mas atualmente insere-se no cotidiano como uma necessidade. Hoje, muitas pessoas afirmam que não conseguem mais viver sem os celulares, outras dizem que através deles podem estar disponíveis 24 horas por dia em qualquer lugar que se encontrem. A popularização do segmento de dispositivos móveis foi rápida, em virtude deste tipo de tecnologia permitir acesso a informações e dados a todo lugar e momento.

Conforme definição do dicionário Aurélio ([Dic 11]), votação é: "Ato ou efeito de votar(-se); Eleição, escrutínio; O conjunto dos votos dados ou recolhidos numa eleição ou assembléia". Uma votação pode ser considerada também com um conceito mais amplo, pode-se chamar de votação qualquer processo de escolha onde, um conjunto de pessoas é responsável por uma decisão final.

Nesse contexto, no processo de realização de uma votação, normalmente são empregados alguns critérios. Um exemplo de critério é a ponderação realizada em determinadas votações, na qual cada participante da votação tem um peso diferente de voto, como por exemplo na eleição para coordenadoria de um curso em que um membro do colegiado tem um peso maior no voto que um estudante de graduação. Um processo de eleição ocorre quando há o desejo de um determinado número de pessoas em escolher uma ou mais pessoas de um grupo (candidatos) que estão concorrendo para representar a sociedade de alguma maneira.

Nos últimos anos foram conduzidas várias experiências para facilitar o processo de eleição. As facilidades foram introduzidas através de novas formas de expressar votos, além das tradicionais baseadas em papel. Exemplos de novas interfaces e sistemas de votação são os touch screens, mensagens SMS de telemóveis e sistemas de votação distribuídos sobre a Internet, conforme definido em trabalho de final de curso dos alunos Luís Miguel Silva Costa e Nuno Alexandre Costa Freta dos Santos ([LUÍ 06], 2006 p. 8).

Esses novos sistemas de votação possuem diversos atrativos, pois possuem características que os sistemas físicos não permitem, como a não necessidade de deslocamento para votar, a possibilidade de boletins para verificar o voto, além de maturidade das pessoas em lidar com as novas tecnologias.

1.1 Contextualização

O Brasil foi um dos pioneiros na criação de urnas eletrônicas, mas muitos países hoje já fazem votação por meio eletrônico, como a Índia, que possui um sistema de votação 100% eletrônica através de um aparelho de votação muito simples em que cada tecla do aparelho representa um candidato. Outros países como Holanda, Estados Unidos e Venezuela utilizam parcialmente sistemas de voto eletrônico.

Figura 1.1: Urna Eletrônica Indiana

Já em países como Reino Unido, Estônia e Suíça o processo de votação eletrônica é ainda mais avançado, sendo possível utilizar votação pela internet. No

Canadá, as eleições para prefeito e nas eleições primárias nos Estados Unidos (algumas cidades) foi possível o uso de celulares, conforme informado pelo site Mundo Tecno [Cyn 10].

Na Suíça, os eleitores recebem em casa uma senha para acessar o sistema de votação eletrônica. Enquanto, na Estônia, a cédula de identidade possui um microchip que pode ser identificado pelo computador, funcionando, assim, semelhante a um certificado digital. Assim eles podem votar em qualquer lugar do mundo sem precisar de cadastramento, enquanto no Brasil não é possível nem mesmo votar pra governador fora da cidade em que se está cadastrado.

Em muito países a urna eletrônica tem muito prestígio, mas nem todos os países querem aderir ao voto eletrônico, alegando falhas na segurança. A Alemanha, por exemplo, estava na fase de testes do sistema de urnas eletrônicas fabricadas na Holanda, mas após a descoberta de falhas na segurança, voltaram a utilizar o papel. Assim como a Alemanha, muitos países hoje não utilizam o voto eletrônico por falta de confiança.

A Austrália tem uma iniciativa de um projeto de votação eletrônica *open source*, na tentativa de criar um projeto de votação transparente e confiável, também esperando com isso que o sistema fique cada vez seguro. As urnas ainda estão em fase de testes mas espera-se que outros países possam copiar a idéia.

1.2 Objetivo

Este trabalho tem como objetivo geral, a análise,o projeto e a elaboração de um protótipo para votação digital em dispositivos móveis que suportem a plataforma Java ME, para o estudo de técnicas de segurança necessários em um sistema de votação.

Outro propósito do trabalho é demonstrar os conhecimentos adquiridos no curso de graduação em Ciência da Computação da Universidade Federal de Santa Catarina, através da elaboração do protótipo desenvolvido e estudo de técnicas e ferramentas de segurança como rede de misturadores e cifragem de mensagens.

1.3 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Estudar conceitos e técnicas de segurança para um sistema de votação digital;
- Incentivar o uso de aplicativos móveis seguros;
- Desenvolver um protótipo de votação para aplicativos móveis;
- Estudar a plataforma J2ME.

1.4 Justificativa

O avanço tecnológico gerou a necessidade de novas formas de desenvolver sistemas de votação eletrônica como meio de expressar a vontade de comunidades de uma sociedade digital, que impossibilita o recurso ao contato físico entre os diversos intervenientes no processo de votação ([SOD 11]).

No final da década de 90, foi inserido no Brasil a votação digital através das urnas eletrônicas sendo o primeiro país do mundo onde todos os eleitores votaram em urnas eletrônicas nas eleições de 2000 ([Wik 11]). O projeto de criação das urnas foi desenvolvido pela *Diebold* ([DIE 11]) em parceria com a fundação CERTI ([FUN 11]). A solução da urna *Diebold* apresentava as características de pouco mais de 8kg de peso, teclado numérico, pequeno monitor de cristal líquido e autonomia de 12 horas de funcionamento sem energia externa.

Em 2000, as urnas receberam ainda um dispositivo de áudio através do qual, usando fones de ouvido, deficientes visuais passaram a ter condições de ouvir a confirmação dos números digitados no teclado, que também contava com identificação em braile.

Em 2010, o censo do IBGE [IBG 11] apurou que a população do Brasil é de 190.755.799 pessoas. E atualmente existem cerca de 212.560.204 celulares no

Brasil [TEL 11]. Considerando esses 2 números, temos uma média de mais de um celular por habitante.

Conforme o apresentado acima fica evidente que há viabilidade da criação de um projeto de votação digital em dispositivos móveis para complementar o atual sistema de urna eletrônica, pois o Brasil está na vanguarda dos países com votação digital dominando tecnologia necessária para executar o projeto.

1.5 Trabalhos Relacionados

Este trabalho apresenta aspectos correlacionados a outros projetos dentre os quais se destacam:

A dissertação de mestrado com o título **Estudo e Implementação de Redes de Comunicação Anônima e aplicação ao Sistema de Votação Digital OSTRACON** [FAB 05] defendida por Fabiano Castro Pereira, que apresenta e implementa uma rede de comunicação anônima utilizada no projeto de Votação Eletrônica OSTRACON.

A dissertação de mestrado de título **Protocolos Criptográficos para Votação Digital** [ARA 02] defendida por Roberto Samarone dos Santos Araújo, que mostra processos e etapas de uma votação segura.

A tese de doutorado de título **On Remote and Voter-Verifiable Voting** [ARA 08] defendida por Roberto Samarone dos Santos Araújo, que trata do voto eletrônico e verificabilidade do voto.

O trabalho de final de curso de título **Uma Ferramenta para obtenção online de certificados digitais para uso de smart cards em aplicações Java Me** [AND 07] do graduado André Luiz Cardoso o qual apresentou uma abordagem sobre o ambiente J2ME.

O trabalho de final de curso de título **MobileREVS - votação electrónica** [LUÍ 06] dos alunos Luís Miguel Silva Costa e Nuno Alexandre Costa Freta dos Santos do Instituto Superior Técnico (MIT Portugal[INS 11]), que apresenta uma abordagem

sobre dispositivos móveis e votação eletrônica.

1.6 Organização do Texto

O capítulo 2 apresenta as ferramentas utilizadas no desenvolvimento do projeto. O capítulo 3 trata da revisão literária do trabalho onde são discutidos aspectos de fundamentos da criptografia, rede de misturadores e votação digital. O capítulo 4 apresenta o protótipo do sistema de votação em dispositivos móveis, abordando detalhes da arquitetura e implementação. O capítulo 5 apresenta a conclusão do trabalho.

Capítulo 2

Levantamento Tecnológico

Neste capítulo serão apresentados as tecnologias e os dispositivos utilizados no desenvolvimento deste protótipo.

2.1 Dispositivos Móveis

A primeira vista, a mobilidade aparentava ser apenas mais uma facilidade, mas atualmente insere-se no cotidiano como uma necessidade. Hoje, muitas pessoas afirmam que não conseguem mais viver sem os celulares, outras dizem que através deles podem estar disponíveis 24 horas por dia em qualquer lugar que se encontrem. A popularização do segmento de dispositivos móveis foi rápida, em virtude deste tipo de tecnologia permitir acesso a informações e dados a todo lugar e momento.

2.2 Java ME

Conforme especificado pela *Sun Microsystems*, a tecnologia Java é basicamente constituída de 3 elementos:

- A linguagem de programação, a qual é utilizada para escrever as aplicações;
- A máquina virtual, com a qual executamos as aplicações;

- Um vasto conjunto de bibliotecas, frameworks e APIs que facilitam o desenvolvimento de aplicações, permitindo o reuso de funcionalidades já implementadas.

Para poder proporcionar o máximo de portabilidade da tecnologia Java, a *Sun Microsystems* dividiu sua tecnologia em 3 edições, listadas por Yuan (2003, apud [AND 07], 2007) da seguinte forma:

1. Java Standard Edition (Java SE): Constitui a base da tecnologia Java. A máquina virtual e as bibliotecas que rodam em computadores pessoais e estações de trabalho são definidas por ela;

2. Java Enterprise Edition (Java EE): Projetada para para rodar aplicações em servidores, essa tecnologia é formada pela Java Standard Edition com várias bibliotecas, APIs, containers e ferramentas específicas;

3. Java Micro Edition (Java ME): Tecnologia projetada para pequenos dispositivos, possuem máquinas virtuais leves e especificamente desenvolvidas para este tipo de dispositivo. Constituída por um subconjunto das bibliotecas da Java SE e por bibliotecas específicas.

Abaixo é exibida a relação entre as edições da família Java:

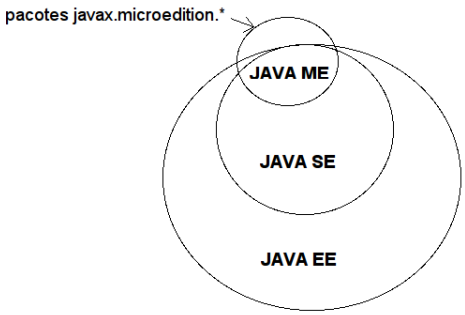

Figura 2.1: Edições da plataforma java

A tecnologia Java ME contém um subconjunto da Java SE, como pode ser observado na ilustração anterior. Embora, apenas restringindo as bibliotecas e APIs e relacionando-as em uma nova edição, não se pode assegurar a portabilidade para pequenos dispositivos.

De acordo com o que foi definido por Ortiz (2004, apud [AND 07], 2007), a tecnologia J2ME não define uma nova linguagem de programação, ela simplesmente adapta a tecnologia Java existente para PDAs e outros dispositivos embarcados. A J2ME mantém a compatibilidade com J2SE quando possível. Para resolver as limitações dos pequenos dispositivos, J2ME às vezes substitui APIs da J2SE e adiciona novas interfaces.

A plataforma J2ME é constituída por um conjunto de APIs padrão, definidas pelo programa Java Community Process [JCP 11] e está hoje em dia presente em milhões de dispositivos (celulares, PDAs, dispositivos automóveis). Isso permite que uma aplicação escrita uma única vez possa ser executada em uma larga variedade de dispositivos.

A Figura 2.1 apresenta uma visão geral da arquitetura da família de plataformas Java, incluindo o J2EE, J2SE, J2ME e Java Cards.

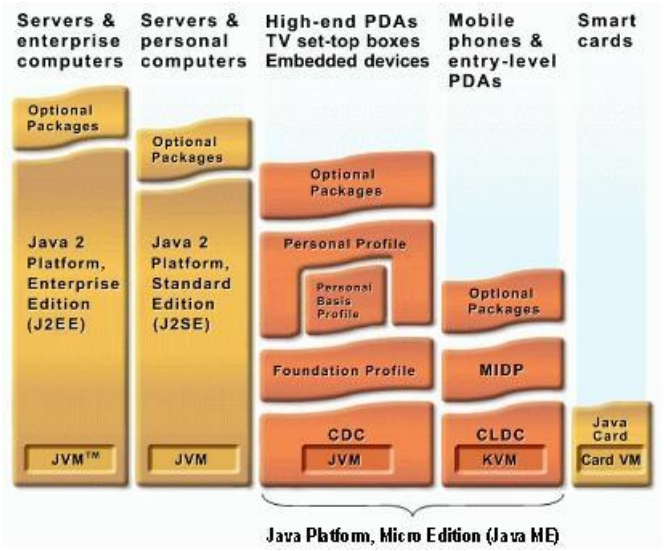

Figura 2.2: Arquitetura do J2ME e relação com o restante família Java

2.2.1 Arquitetura Java ME

A arquitetura da plataforma J2ME é dividida em três camadas: Máquina Virtual, Configurações e Perfis, como mostrado na Figura 2.3.

Figura 2.3: A Arquitetura Java 2 Micro Edition (J2ME)

2.2.2 Configurações

Configuração trata-se de um conjunto de bibliotecas básicas disponíveis para o programador. Ela também define qual o nível de serviços e funcionalidades oferecidos pela máquina virtual. Uma configuração é definida para uma variedade de dispositivos com diferentes aplicações, mas com características em comum. Exemplo: dispositivos com comunicação wireless como celulares, PDA's, pagers, etc. Atualmente existem 2 configurações definidas:

- CLDC (Connected Limited Device Configuration): usado em dispositivos limitados como celulares, PDA's e pagers;

- CDC (Connected Devide Configuration): usado em dispositivos com maior capacidade (processamento e memória) como sistemas de navegação de carros e televisores com conexão à Internet.

Abaixo serão descritas com mais detalhes, as características de cada uma das configurações:

Configuração CLDC

O CLDC foi desenvolvido com o objetivo principal de atender os requisitos de tamanho de memória e recursos computacionais de telefones celulares e outros pequenos dispositivos. Diferentemente da CDC, a CLDC não especifica uma máquina virtual completa, ao invés disso, é especificada uma máquina virtual reduzida, chamada de KVM. Este nome se deve ao fato de seu tamanho ser medido em kilobytes e não em megabytes como nas outras máquinas virtuais conforme definida por Knudsen (2003, apud [AND 07], 2007).

Configuração CDC

A configuração CDC foi projetada para que houvesse suporte a pequenos dispositivos com determinado poder computacional (grande o suficiente para executar uma JVM completa como a da J2SE). Entre os exemplos mais comuns desse tipo de dispositivo estão os set-top boxes de televisão, sistemas de navegação para carros e PDAs com alto processamento Knudsen (2003, apud [AND 07], 2007).

Bibliotecas existentes no Java SE sofrem modificações para suportar restrições de processamento e memória que os pequenos dispositivos possuem. Assim, algumas bibliotecas têm suas interfaces modificadas e outras foram totalmente removidas. O resultado disso é um ambiente de execução Java que se encaixa em dispositivos que tenham no mínimo 2 MB de memória RAM e 2 MB de memória ROM ([Sun 11], 2011).

2.2.3 Perfis

Um perfil especifica o conjunto de bibliotecas para as classes de dispositivos para o qual ele foi projetado. Cada perfil é sempre definido para uma determinada configuração, mas uma configuração pode suportar vários perfis.Os perfis fazem com que a plataforma Java ME apresente opções de APIs para cada categoria de dispositivos, ao invés de definir uma única API que contemple a maioria das situações, como

é feito na plataforma Java SE.

Dois perfis são definidos para o CLDC:

- MIDP (Mobile Information Device Profile);
- IMP (Information Module Profile).

Perfil MIDP

É o primeiro perfil J2ME baseado na configuração CLDC. Muitos aparelhos com suporte ao MIDP estão disponíveis hoje em dia no mercado. Trata-se de uma especificação para o uso da plataforma Java só que feita para dispositivos móveis como celulares, e distribuída também pela *Sun Microsystems*. É o mais conhecido perfil do J2ME baseado na CLDC e KVM, e é o perfil mais desenvolvido e amplamente adotado em muitos lugares por todo o mundo, principalmente em PDAs, telefones celulares e em outros dispositivos móveis.

Perfil IMP

Trata-se de um subconjunto da especificação MIDP 1.0. A principal diferença é que o IMP não faz uso do *Display* utilizado no perfil MIDP e os demais mecanismos de entrada suportados no perfil citado anteriormente. O IMP é basicamente um subconjunto do MIDP e é utilizado em dispositivos com uma interface muito mais limitada.

2.2.4 Máquina Virtual

A máquina virtual define quais as limitações dos programas que executarão no dispositivo. A máquina virtual correspondente ao CLDC é chamada KVM, desenhada especialmente para dispositivos pequenos e com recursos limitados. A KVM mantém os aspectos centrais da Máquina Virtual Java e tem tamanho reduzido.

Para a plataforma Java ME existem também bibliotecas chamadas de pacotes opcionais. Tratam-se de bibliotecas de programação específicas para uma determinada tecnologia. Elas aumentam a capacidade do ambiente, caso estejam sendo usadas no dispositivo.[CRI 11]

2.3 Satsa

De acordo com Ortiz (2005, apud [DAV 09], 2009) a SATSA (Security and Trusted Services API) é uma Optional Package que estende as funcionalidades de segurança para a plataforma J2ME.

Esta extensão adiciona APIs criptográficas, de serviços de assinaturas digitais e de gestão de certificados. Adicionalmente, são definidos dois módulos - SATSA-APDU e SATSA-JCRMI - que, juntamente com extensões à Generic Connection Framework (ver seção anterior), providenciam uma API que permite às aplicações J2ME comunicar com elementos seguros (ver Figura 2.4).

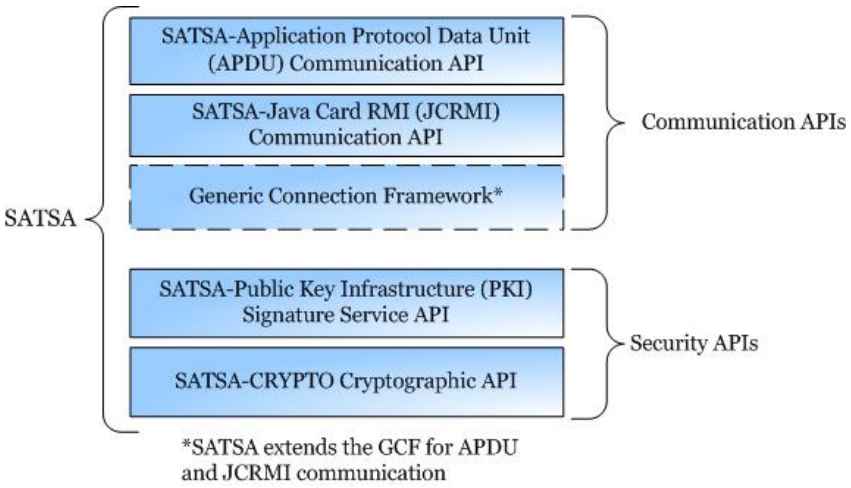

Figura 2.4: APIs do SATSA

2.3.1 Módulos arquiteturais

A SATSA é constituída pelos seguintes quatro módulos, cuja funcionalidade se descreve:

- SATSA-APDU: oferece suporte para a comunicação com as aplicações que estão presentes no smart card;
- SATSA-JCRMI: permite que aplicações J2ME (i.e. MIDlets) efetuem invocações sobre objetos Java a executarem-se no smart card;
- SATSA-PKI: permite efetuar a geração de assinaturas digitais ao nível da aplicação e gestão de certificados;
- SATSA-CRYPTO: é uma biblioteca criptográfica genérica para J2ME.

2.3.2 Modelo de comunicação

O modelo de comunicação do SATSA relaciona-se com a utilização das APIs SATSA-APDU e SATSA-JCRMI. Ao estenderem o J2ME estas suportam a interação das aplicações J2ME do celular, os MIDlets, com aplicações residentes nos smart cards, as applets (ver Figura 2.5). Através desta interação é possível realizar diversos serviços, tais como o armazenamento seguro de dados.

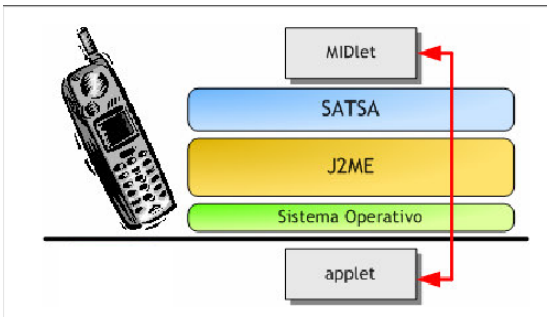

Figura 2.5: Interação dos MIDlets com os applets

A comunicação entre os MIDlets e os applets é baseada num modelo síncrono de pedido/resposta, onde o MIDlet toma o papel de cliente na interação e o applet

toma o papel de servidor. Esta comunicação é realizada através do envio de APDUs (*Application Protocol Data Units*), que correspondem a tramas de dados em bits. Na comunicação com Java Cards este modelo é simplificado através do módulo SATSA-JCRMI, providenciando um modelo orientado a objetos que abstrai a troca dessas tramas.

2.4 Ambiente de desenvolvimento

O desenvolvimento da aplicação Java ME foi realizado no ambiente integrado de desenvolvimento (*Integrated Development Environment - IDE*) de software NetBeans versão 6.9.1, desenvolvido pela *Sun Microsystems*. O desenvolvimento da parte correspondente ao servidor foi realizado na *IDE* Eclipse.

2.4.1 Netbeans

O Netbeans é uma IDE gratuita, de fácil instalação e que roda em diversas plataformas, incluindo Windows, Linux, Mac OS X e Solaris. Essa IDE fornece ferramentas para o desenvolvimento de aplicações para dispositivos móveis que utilizam a linguagem Java.

O Netbeans possibilita a criação, teste, depuração e também a implantação de aplicações desenvolvidas para os perfis MIDP, configurações CLDC e CDC.

Possui uma ferramenta que faz a análise do código, com o intuito de diminuir o tamanho da aplicação identificando componentes que não estão sendo utilizados e verifica a compatibilidade com o perfil MIDP 1.0.

2.4.2 Eclipse

Eclipse é uma IDE (ambiente de desenvolvimento integrado) e um extenso sistema de plugins. É escrito principalmente em Java e pode ser usado para desenvolver aplicações em Java e, por meio de diversos plugins, outras linguagens de programação como Ada, C, C++, Cobol, Perl, PHP e diversas outras.

A IDE Eclipse para desenvolvedores Java EE contém tudo que você precisa para construir aplicações em Java e *Java Enterprise Edition (Java EE)*. Considerado por muitos a melhor ferramenta de desenvolvimento Java disponível. A IDE Eclipse para desenvolvedores Java EE fornece compilação incremental, um editor gráfico de HTML / JSP / JSF, ferramentas de gerenciamento de banco de dados e suporte para os mais populares servidores de aplicações. [WES 11]

2.4.3 Postgres

Como informado na webpage do postgres [POS 11], ferramenta PostgreSQL é um poderoso sistema gerenciador de banco de dados objeto-relacional de código aberto. É um sistema com mais de 15 anos de desenvolvimento e sua arquitetura possui uma forte reputação de confiabilidade, integridade de dados e conformidade a padrões. Uma grande vantagem é o fato de rodar em todos os grandes sistemas operacionais, incluindo GNU/Linux, Unix (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), e MS Windows.

Como um banco de dados de nível corporativo, o PostgreSQL possui funcionalidades sofisticadas como o controle de concorrência multiversionado (MVCC, em inglês), recuperação em um ponto no tempo (PITR em inglês), *tablespaces*, replicação assíncrona, transações agrupadas (savepoints), cópias de segurança a quente (online/hot backup), um sofisticado planejador de consultas (otimizador) e registrador de transações sequencial (WAL) para tolerância a falhas.

Suporta conjuntos de caracteres internacionais, codificação de caracteres multibyte, Unicode e sua ordenação por localização, sensibilidade a caixa (maiúsculas e minúsculas) e formatação. É altamente escalável, tanto na quantidade enorme de dados que pode gerenciar, quanto no número de usuários concorrentes que pode acomodar.

Existem sistemas ativos com o PostgreSQL em ambiente de produção que gerenciam mais de 4TB de dados.

2.4.4 Emuladores Utilizados

O desenvolvimento de aplicações para dispositivos móveis requer a utilização de emuladores de plataformas. As plataformas alvo da ferramenta proposta são Java ME perfil MIDP versão 2.1, configuração CLDC versão 1.1. As seções subsequentes apresentam os emuladores selecionados para o desenvolvimento e teste. Para a plataforma Java ME foi escolhido o emulador *Sun Java Wireless Toolkit*.

Como citado por Davi Garcia Pereira [DAV 09], o kit de desenvolvimento da aplicação escolhido foram: Sun Wireless Toolkit para CLDC versão 2.5.2, utilizando a configuração CLDC com o perfil MIDP.

Trata-se de um conjunto de ferramentas que torna possível criar aplicações para telefones celulares e outros dispositivos sem fio. Apesar do fato de utilizarmos o perfil MIDP 2.1, o Sun Wireless Toolkit para CLDC também suporta muitos pacotes opcionais. Incluindo ambientes de emulação, otimização de desempenho e ajuste de recursos, documentação, e exemplos. O Sun Wireless Toolkit inclui muitos recursos amigáveis ao desenvolvedor. Alguns deles são:

- Suporte às APIs padrão:
    1. CLDC versão 1.1 (JSR 139);
    2. MIDP versão 2.0 (JSR 118);
    3. Wireless Messaging API (JSR 205);
    4. Mobile Media API (JSR 135);
    5. SATSA API (JSR 177); e outras.
- Escolha da interface do emulador;
- Cria projetos de arquivos JAR e JAD;
- Suporta outros emuladores;
- Aplicações de demonstração; e outras.

Capítulo 3

Revisão Literária

3.1 Votação Digital

O conceito de votação é amplo e vai além das conhecidas eleições. Também aplica-se a tudo que envolva a reunião de pessoas para a escolha de algo.

3.1.1 Votação Indireta e Votação Direta

Conforme definiu Araújo ([ARA 02], 2002 p. 11), basicamente existem duas formas principais de votação, a indireta e a direta.

A votação indireta ocorre normalmente em situações na qual lidamos com uma grande quantidade de votantes e muitas questões a serem decididas. Reunir essa grande quantidade de pessoas é um processo de um custo relativamente alto e leva um grande tempo para ser apurado. Em virtude desse fato e de outros que podem surgir, opta-se por se fazer o processo de maneira indireta. Dessa maneira os votantes elegem representantes e estes votam sobre as decisões de interesse dos votantes.

A votação direta ocorre em outras situações onde se tem um número de votantes consideravelmente pequeno. Nesse caso, é mais fácil reunir um mínimo necessário de pessoas para efetuar a votação. Dessa maneira mesmo com um grande número de decisões a serem tomadas, pode-se efetuar a votação com um custo relativamente baixo. Neste tipo de votação os votantes expressam suas opiniões diretamente.

3.1.2 Tipo de Voto

Em uma votação pode-se ter o conceito de voto ponderado na qual são atribuídos diferentes pesos aos votos durante uma tomada de decisão.

Um exemplo de ponderação seria o caso mostrado na dissertação de mestrado de Roberto Araújo ([ARA 02], 2002 p. 12), no qual em uma empresa pode-se ter diversos acionistas e cada um desses possuir um número diferente de ações. Dessa maneira, os votos dos acionistas com maior número de ações teriam um peso maior no resultado final da apuração do que os votos dos acionistas minoritários.

Também existe o conceito no qual não é usada a ponderação, isso é abordado por exemplo nas eleições municipais, nas quais todos os votos têm o mesmo peso.

3.1.3 Atores Participantes de uma Votação Digital

De acordo com Araújo ([ARA 02], 2002 p.12) em votações convencionais, àquelas que usam cédulas em papel e urnas, normalmente existe um conjunto de pessoas com funções específicas. Por exemplo, as pessoas que fiscalizam e administram para que não hajam fraudes ou falhas no processo de votação.

3.1.4 Atores Ativos e Atores Passivos

Os atores do ambiente de votação digital: votante; meio de comunicação e o sistema de votação podem ser atores ativos e atores passivos.

Atores ativos são aqueles que realizam processamento de informações e portanto tem poder computacional e atores passivos são aqueles que não possuem poder computacional. O votante é o ator que vota, porém para que seja possível para o mesmo realizar o ato do voto, é necessário que ele possua software e hardware.

Conforme definido na dissertação de mestrado de Araújo ([ARA 02], 2002 p. 14), denomina-se Processo Cliente de Votação (PCV), o conjunto software/hardware necessário para emitir o voto. Nesse caso estamos assumindo que o PCV e o indivíduo

que utiliza o PCV para votar, são uma entidade única. De maneira semelhante, o conjunto software/hardware do sistema de votação deve cumprir requisitos mínimos para conduzir o processo de votação. Esse conjunto software/hardware descrito anteriormente é denominado Processo Servidor de Votação (PSV).

Tanto o PCV quanto o PSV são atores ativos e precisam realizar computação principalmente relativas à criptografia. O meio de comunicação é um ator passivo pois não precisa realizar nenhum tipo de computação relacionado diretamente ao processo de votação. A função do meio de comunicação é de estabelecer a comunicação entre os atores ativos, para que os mesmos sejam capazes de trocarem informações.

3.1.5 Confiança nos Atores da Votação

Um esquema de votação digital é uma aplicação distribuída constituída por um conjunto de protocolos e mecanismos criptográficos que juntos permitem que uma votação aconteça inteiramente sobre redes de computadores, de maneira segura, mesmo assumindo que seus legítimos participantes possam ter comportamento malicioso, Riera (1999, apud [ARA 02], 2002, p. 15).

Para um funcionamento perfeito de um esquema de votação é necessário que os três principais atores (votante, canal de comunicação e o sistema de votação - PSV) sejam totalmente confiáveis.

Um sistema de votação digital é malicioso se possibilitar que qualquer ator, participante ou não do processo, altere o resultado final da votação. Para resolvermos problemas relativos à maliciosidade de um sistema de votação digital e de todas as entidades envolvidas no processo, é necessário que seja utilizado um protocolo criptográfico de votação digital e que esse atenda a determinados requisitos de segurança. Além de participantes do processo de votação considerados legítimos poderem ser maliciosos, ainda há a possibilidade de que agentes externos tentarem de alguma maneira influir maliciosamente no processo de votação.

3.2 Bouncy Castle

The Legion of Bouncy Castle começou como uma iniciativa para a implementação de um provedor de criptografia para o *Java Cryptographic Extension* (JCE) totalmente gratuito e de código aberto ([AND 07], 2007).

A implementação foi organizada de tal forma que pode ser empacotada tanto para Java SE como para Java ME. Por ser um projeto de código aberto, esta biblioteca apresenta uma série de vantagens, enumeradas por Yuan (2003, apud [AND 07], 2007) na seguinte lista:

- Bugs ou falhas de segurança são corrigidos rapidamente;
- As APIs foram projetadas de forma flexível, e juntamente com o modelo de desenvolvimento comunitário, permitem que qualquer desenvolvedor possa contribuir com novas implementações de algoritmos. Consequentemente, a biblioteca implementa um grande número de algoritmos conhecidos;
- A comunidade de desenvolvedores está sempre otimizando as implementações existentes, garantindo a melhoria do desempenho das operações criptográficas;
- É gratuita.

Esta biblioteca suporta uma diversidade muito grande de algoritmos, sendo bem completa e funcional, apresentando muito mais opções de algoritmos e operações criptográficas (hash, por exemplo) quando comparada com os pacotes opcionais da SATSA (Pacote [*Optional Package*] que estende as funcionalidades de segurança para a plataforma J2ME). Porém, ao contrário da SATSA, não é padronizada como pacotes opcionais, não sendo, portanto, suportada nativamente pelos dispositivos, o que ocasiona um aumento significativo do tamanho da aplicação Java ME, pois esta tem que incluir em seu pacote as classes da Bouncy Castle utilizadas.

Outra desvantagem é o fato de não dar suporte ao uso de smart cards, além de carecer de uma documentação eficaz, principalmente no âmbito do pacote para Java ME (lightweight).

3.3 Fundamentos de Criptografia

3.3.1 Introdução

A palavra criptografia surgiu da fusão das palavras gregas “kryptós” que significa “oculto” e “gráphein” que significa “escrever”. A criptografia trata-se de um conjunto de conceitos e técnicas que tem por objetivo codificar uma informação de forma que somente o emissor e o receptor possam ter acesso a mesma, dessa forma evita que um intruso qualquer consiga interpretá-la. Para isso, uma série de técnicas são usadas e muitas outras surgem com o passar do tempo. [EME 05]

A mensagem original emitida é determinada de texto aberto. O texto aberto é processado por um algoritmo de criptografia tornando-se um texto cifrado. O processo de obtenção desse texto aberto original a partir do texto cifrado é denominado decifragem. [ARA 02]

3.3.2 Criptografia Simétrica

Os algoritmos que se utilizam somente de uma chave tanto para o processo de cifragem quanto para o de decifragem são chamados de algoritmos criptográficos simétricos. Esses algoritmos foram os primeiros a serem criados e são muito utilizados até hoje para garantir o sigilo de informações. [DAV 09]

O funcionamento desse algoritmo se baseia no fato de ambos, transmissor e receptor de alguma mensagem compartilharem uma mesma chave. Ou seja, uma mensagem só pode ser decifrada pela mesma chave que foi utilizada para cifra-lá.

Sendo assim, fica fácil perceber que se receptor e transmissor, ambos tiverem a chave e não a compartilham com ninguém, somente eles podem entender a mensagem garantindo assim a privacidade e a autenticidade. Mas esse método apresenta uma desvantagem que é a troca da chave. Uma tarefa difícil de manter o sigilo em um grande ambiente como a Internet.

Abaixo está uma ilustração de como é realizada a criptografia simétrica:

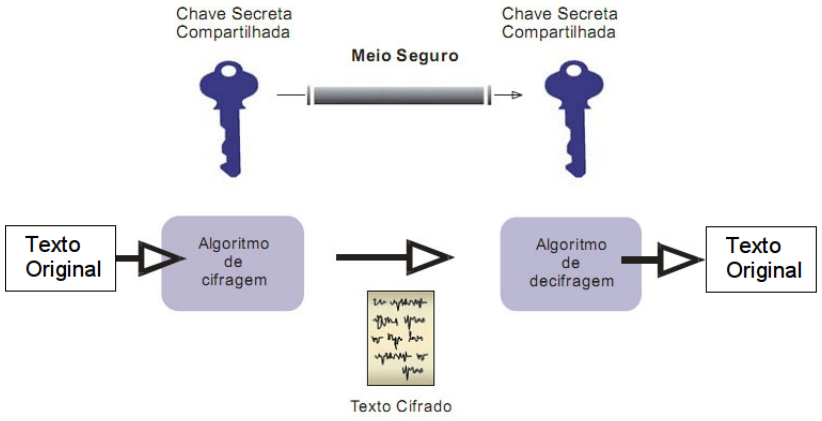

Figura 3.1: Ilustração do funcionamento da criptografia simétrica.

3.3.3 Criptografia Assimétrica

Também chamada de criptografia de chave pública, esse tipo de criptografia proposto por Diffie e Hellman, é um sistema no qual é utilizado um par de chaves distintas tanto para cifrar quanto para decifrar. Normalmente uma delas é mantida em segredo, e recebe o nome de chave privada (KR_a) e a outra é amplamente divulgada, sendo denominada chave pública (KU_a). O funcionamento ocorre da seguinte maneira: caso se cifre a mensagem com a chave privada, ela só será decifrada pela chave pública e vice-versa.

Sendo assim, E_{KU_A} consiste em aplicar o algoritmo de cifragem utilizando a chave pública de uma entidade "A", e D_{KR_A} resume-se na aplicação do algoritmo para decifrar a mensagem utilizando a chave privada de "A". Isso considerando que deve haver uma forte relação entres as duas chaves citadas, de maneira a garantir que D_{KR_A} (E_{KU_A}(x)) = x. Como exemplos de cifradores assimétricos, podemos citar os algoritmos RSA e ElGamal.

A criptografia assimétrica não substitui a criptografia simétrica, pois os algoritmos utilizados são lentos e vulneráveis a alguns ataques. Normalmente se utiliza a criptografia de chave pública para distribuição com segurança de chaves simétricas (chaves de seção), pois estas serão utilizadas para a cifragem das mensagens.

Essa técnica funciona da seguinte maneira: o transmissor gera uma chave

simétrica (chave de seção), esta é cifrada usando a chave pública do receptor (chave assimétrica), depois este a envia, e apenas quem possuir a chave do receptor poderá decifrar o conteúdo enviado. Recuperando assim a chave simétrica que então poderá ser usada para a comunicação. Esse modelo, também chamado de criptografia híbrida, aproveita as vantagens dos 2 modelos (simétrico e assimétrico). A seguir há uma figura ilustrando essa técnica:

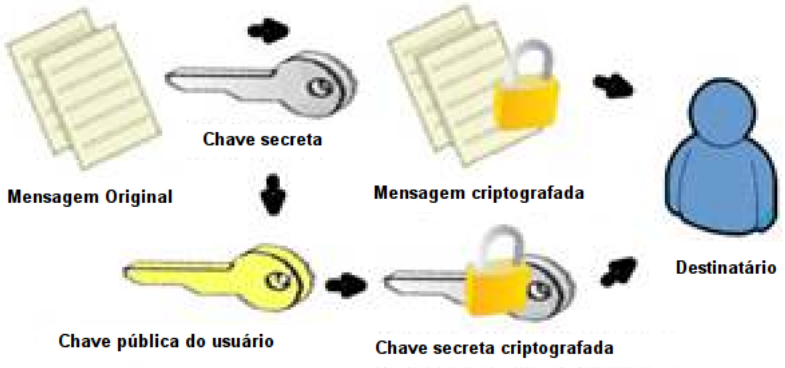

Figura 3.2: Processo de criptografia e distribuição da chave secreta na criptografia híbrida.

3.4 ElGamal

Na criptografia, o sistema de criptografia ElGamal é um algoritmo de encriptação de chaves assimétricas baseada na troca de chaves de Diffie-Hellman. Foi descrita em 1984 por Taher ElGamal. [WEE 11]

Este algoritmo permite confirmar a autenticidade de uma mensagem enviada, mesmo que tenha sido enviada em um canal não seguro, além do fato de uso do problema de logaritmo de algoritmo discreto. A geração de suas chaves segue o padrão das chaves públicas, ou seja, cada entidade gera um par de chaves e acertam a forma como vão distribuir as chaves públicas.

A criptografia ElGamal consiste em três componentes: A geração de chaves, a cifragem, e a decifragem:

Geração das Chaves

A entidade “A” procede do seguinte modo:

- Gera de forma aleatória um número primo p de grande dimensão e g, que é uma raiz primitiva módulo p;
- Seleciona aleatoriamente um inteiro a, onde $1 \leq a \leq p$-2 e calcula $r = g^a \pmod{p}$;
- A chave pública da entidade será (p, g, r), e sua chave privada será a.

Cifragem

- A entidade B obtém a chave pública de “A” (p, g, r);
- Representa a mensagem como se m fosse um inteiro no intervalo 0, 1, ..., p-1;
- Seleciona aleatoriamente um inteiro k, $1 \leq k \leq p$-2;
- Calcula $\psi = g^k \pmod{p}$ e $\delta = m \times r^k \pmod{p}$;
- O texto cifrado será $c = (\psi, \delta)$.

Decifragem

Para haver a recuperação, “A” deve:

- Usando a chave privada a, calcular $\psi^{(p-1-a)} \pmod{p}$;
- Recuperar a mensagem m calculando $\psi^{(-a)} \times \delta \pmod{p}$.

O problema da cifra ElGamal é que ele possui um valor de expansão de mensagem cifrada de 2, ou seja, o comprimento da mensagem cifrada é o dobro da mensagem clara. O uso de aleatoriedade em seu código busca aumentar sua segurança ao evitar a eficiência dos ataques através da análise estatística.

A segurança do ElGamal está baseada na dificuldade em se tirar logaritmos discretos de números grandes, ou seja, de se obter *a* a partir de β, *p* e α (*a* = $\log_\alpha \beta$ mod p). α e β devem ser grandes, além disso, deve-se escolher um *k* diferente para cada *m*. [ALI]

3.5 Rede de Misturadores

Como definiu Araújo ([ARA 02], 2002, pag 25), sempre que se deseja uma não ligação entre emissor e receptor em uma determinada comunicação, uma rede de misturadores é uma ferramenta bastante apropriada. Em síntese, um misturador além de encaminhar as mensagens que chegam, também esconde a relação entre as mensagens que nele chegam e que dele saem. Isso acontece devido ao fato de o misturador, agrupar um determinado número de mensagens que chegam para ele e misturá-las antes que sejam encaminhadas.

Tendo como intuito não correlacionar as mensagens que chegam com as mensagens que saem, todas as mensagens que serão embaralhadas precisam serem cifradas. De modo que cifragens sucessivas da mesma mensagem mostram diferentes resultados. É imprescindível também que todas as mensagens possuam um tamanho uniforme, para que não seja possível estabelecer uma relação entre a mensagem e o seu respectivo tamanho. Tal propriedade pode ser solucionada através da adição de "enchimentos" aleatórios.

Para um grupo de usuários se comunicando, a não ligação entre as mensagens dos mesmos é garantida, exceto para o próprio misturador. Tendo como objetivo melhorar os resultados, ou seja, obter mais segurança, uma rede de misturadores é montada. Tal rede é montada a partir do cascateamento de vários misturadores.

Uma rede de misturadores envelopa as mensagens para que essas sejam lidas apenas por misturadores dirigidos. Envelopes digitais são abertos e o conteúdo obtido é misturado antes de ser encaminhado para o misturador seguinte que atuará da mesma forma. A entrega da mensagem ao destino é feito pelo último misturador do

cascateamento, logo após efetuar a última mistura.

Utilizando uma rede de misturadores, somente um único misturador precisa ser honesto para prevenir os outros misturadores de não quebrar a ligação entre origem e destino. O protocolo Farnel (2001, apud [ARA 02], 2002, p. 24) é um exemplo de protocolo de votação digital segura que utiliza este recurso criptográfico.

Abaixo é apresentada uma figura que ilustra o funcionamento da rede:

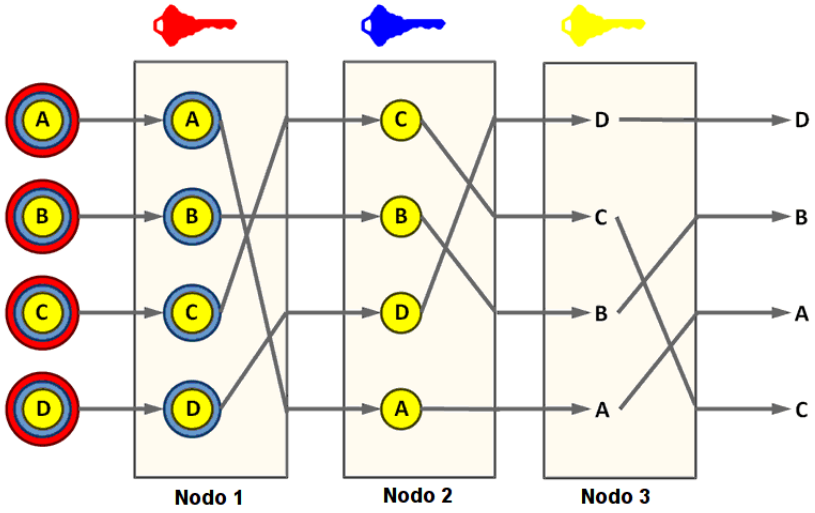

Figura 3.3: Ilustração mostrando embaralhamento das mensagens em cada um dos nodos da rede.

Capítulo 4

Protótipo do Sistema de Votação em Dispositivos Móveis

Neste capítulo será apresentado o protótipo de um sistema de votação para dispositivos móveis e suas funcionalidades, mostrando uma visão geral do sistema com os requisitos, casos de uso e diagramas necessários alem dos fluxos de trabalhos. Também serão apresentados os resultados obtidos na votação programada.

4.1 Preparação do ambiente

Antes de iniciar uma eleição é necessário preparar o ambiente e as configurações para a votação. Um dos principais requisitos de segurança é a comunicação segura entre o cliente (votante) e o servidor que armazena os votos, para isso foi utilizado uma implementação da rede de misturadores de Chaum, desenvolvida pelo Laboratório de Segurança em Computação da Universidade Federal de Santa Catarina e denominado *JMixNet*.

A rede de mistura foi criada em um ambiente virtual com 3 nodos (servidores/misturadores) em decorrência de simplicidade e também devido a dificuldade da alocação de um maior número de servidores. A criação de cada servidor da rede de misturadores da *jmixnet*, assim como a forma de inicializar a rede pode ser encontrada

no apêndice B deste trabalho .

Os outros 2 módulos importantes no trabalho são o servidor, responsável por guardar os dados da eleição e o cliente (ambiente que realização a votação propriamente dita):

- O ambiente do servidor, necessita apenas de um certificado assinado pela autoridade certificadora da *JMixNet*, para rodar o módulo servidor *"JmixnetServer"* da *jmixnet* além do java jdk instalado. A instalação do certificado assinado pode ser encontrada no apêndice B, assim como a instalação do jdk também pode ser encontrada no apêndice B;

- O ambiente onde foi desenvolvido o aplicativo cliente necessita apenas do J2ME instalado, que pode ser encontrado no apêndice B.

4.1.1 Segurança do Sistema

Este item irá tratar dos requisitos de segurança do sistema. A segurança do protótipo de votação digital implementado é baseado em dois itens principais. A rede de misturadores que é a responsável por não alinhar o votante ao servidor que armazena os votos e a cifra ElGamal utilizada nos votos para não permitir que os votos possam ser visualizados quando salvos no banco.

Cifrar com ElGamal

A Cifra ElGamal, possui uma característica muito importante, a capacidade de gerar uma cifra diferente para uma mesma mensagem utilizando-se de uma mesma chave.

Essa característica importante foi decisiva na escolha do algoritmo ElGamal no protótipo. O mais interessante desse algoritmo é o fato de que um votante é capaz de receber a chave pública ElGamal do servidor e cifrar seu voto com essa chave, e uma pessoa com acesso a base de dados não é capaz de descobrir se o voto foi para candidato A ou B. Isso decorre de que, caso seja feita cifragem a partir de um

texto, como por exemplo: um código de candidato, não há como ser obtido um mesmo resultado de cifra para o mesmo texto, e por consequência não há como usar isso como um critério de comparação.

No protótipo desenvolvido, o servidor é o responsável por gerar o par de chaves ElGamal, para cada eleição é gerado um par de chaves. A chave pública fica disponível para download a qualquer momento. O download da chave é efetuado automaticamente após ser concluído o login via celular.

Nesse contexto, a chave pública é utilizada no processo de cifra dos votos, e a chave privada fica armazenada no servidor para ser utilizada para decifrar e por consequência, apurar o resultado da votação assim que o prazo final da eleição é atingido. Para geração das chaves foi utilizado o conjunto de APIs *BouncyCastle*, conforme código mostrado abaixo:

```
1  private KeyPair geraParDeChaves_elgamal(int tamanhoChave)
2  throws Exception{
3          Security.addProvider(
4          new org.bouncycastle.jce.provider.BouncyCastleProvider());
5          KeyPairGenerator generator = KeyPairGenerator.getInstance(
6          "ElGamal", "BC");
7          SecureRandom random = new SecureRandom();
8          generator.initialize(tamanhoChave, random);
9          KeyPair pair = generator.generateKeyPair();
10       return pair;
11     }
```

E para a decifragem foi utilizado o código mostrado abaixo:

```
1  public byte[] decifraTexto(byte[] textoCifrado, Key chavePrivada)
2      throws Exception{
3          Security.addProvider(
4      new org.bouncycastle.jce.provider.BouncyCastleProvider());
5          byte[] input = textoCifrado;
6          Cipher decifrador = Cipher.getInstance("ElGamal/None/NoPadding", "BC");
7          decifrador.init(Cipher.DECRYPT_MODE, chavePrivada);
8          byte[] plainText = decifrador.doFinal(input);
9          return plainText;
10     }
```

No cliente foi utilizado o conjunto de APIs *BouncyCastle* reduzida, mais

compacta, possibilitando assim ser executada num ambiente J2ME.

Rede de Misturadores

A rede de misturadores desempenha um papel importante na segurança do sistema. Sua principal função é não permitir que exista uma ligação direta entre o votante e o servidor que armazena os votos. O votante envia o seu voto diretamente para o primeiro servidor na rede de misturadores através de um cliente *jmixnet*, como mostrado no código abaixo:

```
1       protected void configureClient() throws JMixNetException {
2           //ensure Bouncy Castle Provider is available
3           if (Security.getProvider("BC") == null) {
4               Security.addProvider(new BouncyCastleProvider());
5           }
6           //message config
7           messageSize = config.getMessageSize();
8           dataBuffer = ByteBuffer.allocate(messageSize);
9           tempBuffer = ByteBuffer.allocate(messageSize);
10          try {
11              //server certificates
12              serverCerts = (ArrayList)config.getServerCerts();
13              Collections.reverse(serverCerts);
14          }
15          catch (ConfigException ex) {
16              showError("Unable to get server certificates:\n"
17          + ex.getMessage());
18              throw new JMixNetException(ex);
19          }
20          //calculate max data size the client can send
21          iterator = serverCerts.iterator();
22          int totalKeyLength = 0;
23          while (iterator.hasNext()) {
24              totalKeyLength += X509CertificateInfo.getKeyBytesSize(
25          (X509Certificate)iterator.next());
26          }
27          maxDataSize = messageSize - hashFieldSize - totalKeyLength
28      - dataDescSize;
29          try {
30              //cryptography services and hash
31              symCrypto = new SymCryptographer();
32              asymCrypto = new AsymCryptographer();
```

```
33              digester = MessageDigest.getInstance("MD5");
34          }
35          catch (CryptoException ex) {
36              showError("Unable to use cryptography services:\n"
37          + ex.getMessage());
38              throw new JMixNetException();
39          }
40          catch (NoSuchAlgorithmException ex) {
41              showError("Unable to get a message digest instance:\n"
42          + ex.getMessage());
43              throw new JMixNetException();
44          }
45      }
```

```
1   public JMixNetClient() throws JMixNetException {
2           try {
3               config = new JMixNetConfigManager();
4           }
5           catch (ConfigException ex) {
6               showError("Config error occurred:\n" + ex.getMessage());
7               throw new JMixNetException();
8           }
9           configureClient();
10      }
```

Recebido o voto na rede, o primeiro servidor se encarrega de armazenar o voto por um período de tempo pré-determinado ou até acumular uma determinada quantidade de votos, os embaralha e encaminha ao próximo servidor.

Os servidores sequentes na rede, tem o mesmo comportamento, guardam os votos, os embaralham e os enviam ao próximo nodo da rede. Esse processo ocorre até que se alcance o último nodo da rede, que envia os votos ao servidor que possui um receptor de mensagens ativo, ou seja, há uma porta aberta aguardando sempre que um novo voto chegue, como mostrado no código abaixo:

```
1   public class ReceiverServer {
2       public static void main(String[] args) {
3           ServerSocketFactory factory =
4       ServerSocketFactory.getDefault();
5           Socket clientCon;
6           InputStream cliInput;
7           byte[] msgBuf = new byte[4096];
```

```
8           ServerSocket server = null;
9           int amountRead;
10          System.out.println("Receiver server running.");
11          try {
12              server = factory.createServerSocket(2000);
13          }
14          catch (IOException e) {
15              e.printStackTrace();
16          }
17          while (true) {
18              try {
19                  clientCon = server.accept();
20                  cliInput = clientCon.getInputStream();
21                  while ((amountRead = cliInput.read(msgBuf)) > 0) {
22                  }
23                  clientCon.close();
24              }
25              catch (IOException e1) {
26                  e1.printStackTrace();
27              }
28          }
29      }
30  }
```

4.2 Protótipo Votação

Para o projeto da aplicação foram definidos os requisitos, a arquitetura, os casos de uso e diagramas UML apresentados nas seções subsequentes, a última seção trata da apresentação desenvolvida.

Requisitos Funcionais Cliente

- O software deve poder ser usado em uma ou mais eleições;
- O software deve fazer uso de uma rede de misturadores;
- O software deve ser compatível com a tecnologia MIDP 2.0;
- O software deve ser compatível com a tecnologia CLDC;

- O software devera usar a plataforma Java ME.

Requisitos Não Funcionais Cliente

- O tempo de resposta do sistema não deve ultrapassar 30 segundos;
- O software do sistema deve ser operacionalizado sobre qualquer sistema operacional que suporte java (Padro j2me);
- O usuário deverá fornecer informações como login e senha para autenticar-se;
- O usuário do sistema (módulo cliente), deve possuir um dispositivo com suporte a java, através do qual irá realizar votação;
- O voto deve ser cifrado usando o PIN (*Personal Identification Number* - Número de Identificação Pessoal) e a chave pública importada do servidor;
- O cliente deverá ler arquivos no formato de arquivo xml;
- O cliente deverá ser capaz de cifrar votos utilizando o algoritmo de encriptação El-Gamal.

Requisitos Funcionais Servidor

- O software deve possibilitar a realização de uma ou mais votações;
- Os usuários devem poder visualizar o resultado da apuração de votos;
- Banco de dados Relacional;
- Navegador web;
- Sistema operacional.

Requisitos Não Funcionais Servidor

- A base de dados deve ser protegida para acesso apenas de usuários autorizados;
- O tempo de resposta do sistema não deve ultrapassar 30 segundos;
- O software do sistema deve ser operacionalizado sobre qualquer sistema operacional que suporte java (Padrão J2SE);
- Acessível via web, utilizando páginas jsp;
- O usuário deverá fornecer informações como login e senha para autenticar-se;
- O cliente deverá ser capaz de cifrar seus votos utilizando o algoritmo de cifragem El-Gamal;
- O software deverá ser capaz de gerar pares de chaves com o algoritmo El-Gamal.

4.2.1 Arquitetura

O desenvolvimento da aplicação foi baseado na comunicação cliente (aplicativo móvel) - rede de misturadores - servidor , onde o cliente se comunica com a rede de misturadores através do cliente da rede de mistura, enviando o voto para a rede e a rede de misturadores envia o voto para o servidor *jmixnet* que recebe os votos e os envia para o servidor final que armazena os votos. Os detalhes da arquitetura de cada parte do protótipo pode ser vista abaixo.

4.2.1.1 Cliente - Dispositivo Móvel

A arquitetura utilizado no cliente foi o *MVC - Model-View-Controller* que divide a aplicação em 3 partes básicas:

- Modelo - parte responsável pela lógica da aplicação, onde fica a parte do código que transforma os dados de entrada em resultados;
- Controle - parte responsável por ligar a lógica à interface;

- Visão - parte responsável pela interface com o usuário.

4.2.1.2 Misturadores

Os misturadores da rede tem a função de receber as mensagens(votos) guardá-los por um período de tempo, embaralhá-los e então enviá-los ao próximo misturador da rede ou ao destino final, caso seja o último misturador. O objetivo por trás disso é quebrar a ligação que há entre quem envia e quem recebe uma mensagem, ou seja, retirar a ligação do votante com o servidor que armazena os votos, garantindo assim maior segurança ao votante.

A rede de misturadores implementada para o projeto foi criada em um ambiente virtual com 3 nodos (misturadores). O misturador 1 é responsável por receber as mensagens dos clientes (aplicativos móveis). Todos os votos enviados dos dispositivos móveis são direcionados somente para o misturador 1, através do *JMixNetClient* (cliente da rede de misturadores) que está embutido dentro da aplicação cliente(dispositivo móvel que irá votar). Após guardar por um tempo e acumular determinada quantidade de mensagens, o misturador as embaralha e as envia para o segundo misturador, que por sua vez realiza o mesmo processo (acúmulo de mensagens e embaralhamento) e em seguida envia para o misturador 3. O misturador 3 é o último nodo da rede, responsável por enviar os votos para o servidor final, que será responsável por armazenar os votos.

Os detalhes da implementação da rede de misturadores *jmixnet* pode ser encontrada na dissertação de mestrado de Fabiano Castro Pereira ([FAB 05], 2005).

4.2.1.3 Servidor

O Servidor que armazena os votos, assim como o cliente, trabalha no padrão *MVC - Model-View-Controller*, já explicado anteriormente.

A seguir está ilustrada a arquitetura do sistema:

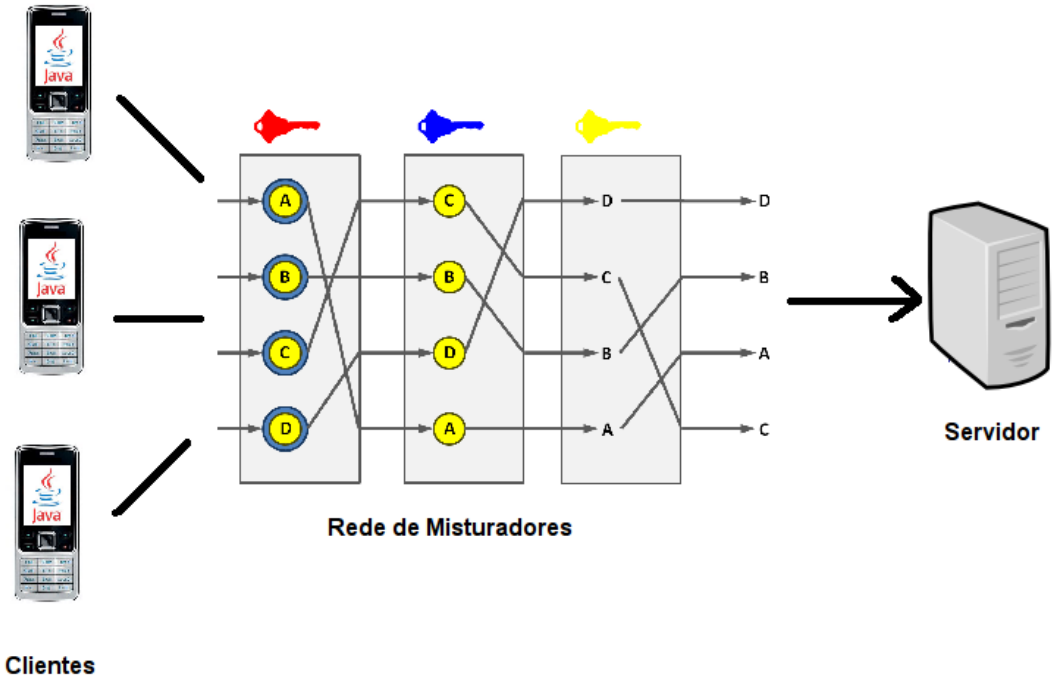

Figura 4.1: Arquitetura do Sistema

4.2.2 Casos de Uso

Os casos de uso descritos a seguir foram elaborados a partir dos requisitos descritos anteriormente.

4.2.2.1 Casos de Uso Servidor

Cadastrar

Cadastrar Eleição

1. O gerente do sistema acessa no menu a opção de cadastro e seleciona o item *Cadastro de Eleição*.
2. O sistema exibe o formulário de cadastro de eleição, onde alguns campos com asteriscos são considerados de caráter obrigatório: *nome*, *data inicial*, *data final*.
3. O gerente do sistema terá acesso aos botões: *salvar* e *cancelar*.

4. O gerente do sistema preenche os dados.

5. O gerente do sistema clica no botão *cancelar*.

6. O gerente do sistema clica no botão *salvar*.

Tratamento de exceções:

- 6a. Já existe uma eleição cadastrada com esse nome.

    1. O sistema irá voltar a tela de cadastro para que o gerente do sistema modifique o nome da eleição a ser cadastrada.
    2. O gerente do sistema digita um novo nome de eleição.
    3. Retorna ao fluxo principal no passo 4.

- 6b. O período escolhido (data inicial e data final) são incompatíveis.

    1. O sistema exibe uma mensagem para o gerente de que o período selecionado é incompatível (data inicial posterior a final ou datas anteriores a data corrente).
    2. O sistema irá voltar a tela de cadastro para que o gerente do sistema modifique o período da eleição a ser cadastrada.
    3. O gerente do sistema seleciona um novo período (data inicial ou final).
    4. Retorna ao fluxo principal no passo 4.

Cadastrar Disputa

1. O gerente do sistema acessa no menu a opção de cadastro e seleciona o item *Cadastro de Disputa*.

2. O sistema exibe o formulário de cadastro de disputa, onde alguns campos com asteriscos são considerados de caráter obrigatório: *nome*.

3. O gerente do sistema terá acesso aos botões: *salvar* e *cancelar*.

4. O gerente do sistema preenche os dados.
5. O gerente do sistema clica no botão *cancelar*.
6. O gerente do sistema clica no botão *salvar*.

Tratamento de exceções:

- 6a. Já existe uma disputa cadastrada com esse nome.
    1. O sistema irá voltar a tela de cadastro para que o gerente do sistema modifique o nome da disputa a ser cadastrada.
    2. O gerente do sistema digita um novo nome de disputa.
    3. Retorna ao fluxo principal no passo 4.

Cadastrar Candidato

1. O gerente do sistema acessa no menu a opção de cadastro e seleciona o item *Cadastro de Candidato*.
2. O sistema exibe o formulário de cadastro de candidato, onde alguns campos com asteriscos são considerados de caráter obrigatório: *nome*, *código*, *foto*, *disputa*.
3. O gerente do sistema terá acesso aos botões: *salvar* e *cancelar*.
4. O gerente do sistema preenche os dados.
5. O gerente do sistema clica no botão *cancelar*.
6. O gerente do sistema clica no botão *salvar*.

Tratamento de exceções:

- 6a. Já existe um candidato cadastrado com esse código para essa disputa.
    1. O sistema irá voltar a tela de cadastro para que o gerente do sistema modifique o código do candidato a ser cadastrado.

2. O gerente do sistema digita um novo código de candidato.
3. Retorna ao fluxo principal no passo 4.

Cadastrar Votante

1. O gerente do sistema acessa no menu a opção de cadastro e seleciona o item *Cadastro de Votante*.
2. O sistema exibe o formulário de cadastro de votante, onde alguns campos com asteriscos são considerados de caráter obrigatório: *nome*, *identificação*, *tipo de identificação*, *login*, *senha*, *certificado digital*.
3. O gerente do sistema terá acesso aos botões: *salvar* e *cancelar*.
4. O gerente do sistema preenche os dados.
5. O gerente do sistema clica no botão *cancelar*.
6. O gerente do sistema clica no botão *salvar*.

Tratamento de exceções:

- 6a. Já existe uma votante cadastrado com essa identificação para esse tipo de identificação.
    1. O sistema irá voltar a tela de cadastro para que o gerente do sistema modifique o dado no campo identificação do votante a ser cadastrado.
    2. O gerente do sistema digita um novo dado para o campo identificação do votante.
    3. Retorna ao fluxo principal no passo 4.
- 6b. Já existe uma votante cadastrado com esse login.
    1. O sistema irá voltar a tela de cadastro para que o gerente do sistema modifique o dado no campo login do votante a ser cadastrado.

2. O gerente do sistema digita um novo dado para o campo login do votante.
3. Retorna ao fluxo principal no passo 4.

- 6c. Já existe um votante cadastrado com esse certificado digital.

1. O sistema irá voltar a tela de cadastro para que o gerente do sistema faça o upload de outro certificado digital para o votante a ser cadastrado.
2. O gerente do sistema faz o upload de outro certificado digital para o campo certificado digital do votante.
3. Retorna ao fluxo principal no passo 4.

Remover

Remover Eleição

1. O gerente do sistema acessa no menu a opção de administração e seleciona o item *Remover Eleição*.
2. O sistema gera uma listagem de eleições e um campo check ao lado de cada eleição.
3. O gerente do sistema terá acesso aos botões: *remover* e *cancelar*.
4. O gerente do sistema seleciona a eleição que deseja remover.
5. O gerente do sistema clica no botão *cancelar*.
6. O gerente do sistema clica no botão *remover*.
7. O sistema exibe os dados da eleição numa nova tela, na qual exibe 2 botões, para que o usuário confirme a exclusão.
8. Após confirmada a exclusão, todos os dados pertencentes a essa eleição (disputas, candidatos, votantes cadastrados) são excluídos.

Tratamento de exceções:

- 6a. O usuário não selecionou nenhuma eleição.

    1. O sistema irá voltar a tela de remoção para que o gerente do sistema selecione uma eleição que deseja remover.
    2. Retorna ao fluxo principal no passo 4.

Remover Disputa

1. O gerente do sistema acessa no menu a opção de administração e seleciona o item *Remover Disputa*.
2. O sistema gera uma listagem de disputas e um campo check ao lado de cada disputa.
3. O gerente do sistema terá acesso aos botões: *remover* e *cancelar*.
4. O gerente do sistema seleciona a disputa que deseja remover.
5. O gerente do sistema clica no botão *cancelar*.
6. O gerente do sistema clica no botão *remover*.
7. O sistema exibe os dados da disputa numa nova tela, na qual exibe 2 botões, para que o usuário confirme a exclusão.
8. Após confirmada a exclusão, todos os dados pertencentes a essa disputa são excluídos.

Tratamento de exceções:

- 6a. O usuário não selecionou nenhuma disputa.

    1. O sistema irá voltar a tela de remoção para que o gerente do sistema selecione uma disputa que deseja remover.
    2. Retorna ao fluxo principal no passo 4.

Remover Candidato

1. O gerente do sistema acessa no menu a opção de administração e seleciona o item *Remover Candidato*.

2. O sistema uma gera listagem de disputas e um campo check ao lado de cada candidato.

3. O gerente do sistema terá acesso aos botões: *remover* e *cancelar*.

4. O gerente do sistema seleciona o candidato que deseja remover.

5. O gerente do sistema clica no botão *cancelar*.

6. O gerente do sistema clica no botão *remover*.

7. O sistema exibe os dados do candidato numa nova tela, na qual exibe 2 botões, para que o usuário confirme a exclusão.

8. Após confirmada a exclusão, todos os dados pertencentes a esse candidato são excluídos.

Tratamento de exceções:

- 6a. O usuário não selecionou nenhum candidato.

    1. O sistema irá voltar a tela de remoção para que o gerente do sistema selecione um candidato que deseja remover.

    2. Retorna ao fluxo principal no passo 4.

Remover Votante

1. O gerente do sistema acessa no menu a opção de administração e seleciona o item *Remover Votante*.

2. O sistema gera uma listagem de disputas e um campo check ao lado de cada votante.

3. O gerente do sistema terá acesso aos botões: *remover* e *cancelar*.

4. O gerente do sistema seleciona o candidato que votante que deseja remover.

5. O gerente do sistema clica no botão *cancelar*.

6. O gerente do sistema clica no botão *remover*.

7. O sistema exibe os dados do votante numa nova tela, na qual exibe 2 botões, para que o usuário confirme a exclusão.

8. Após confirmada a exclusão, todos os dados pertencentes a esse votante são excluídos.

Tratamento de exceções:

- 6a. O usuário não selecionou nenhum votante.

    1. O sistema irá voltar a tela de remoção para que o gerente do sistema selecione um votante que deseja remover.
    2. Retorna ao fluxo principal no passo 4.

4.2.3 Diagramas

A seguir são apresentados os casos separados entre casos de uso do Administrador/gerenciador do sistema, responsável por criar a eleição e configurá-la, e os casos de uso do votante.

4.2.3.1 Diagrama de casos de uso do Administrador

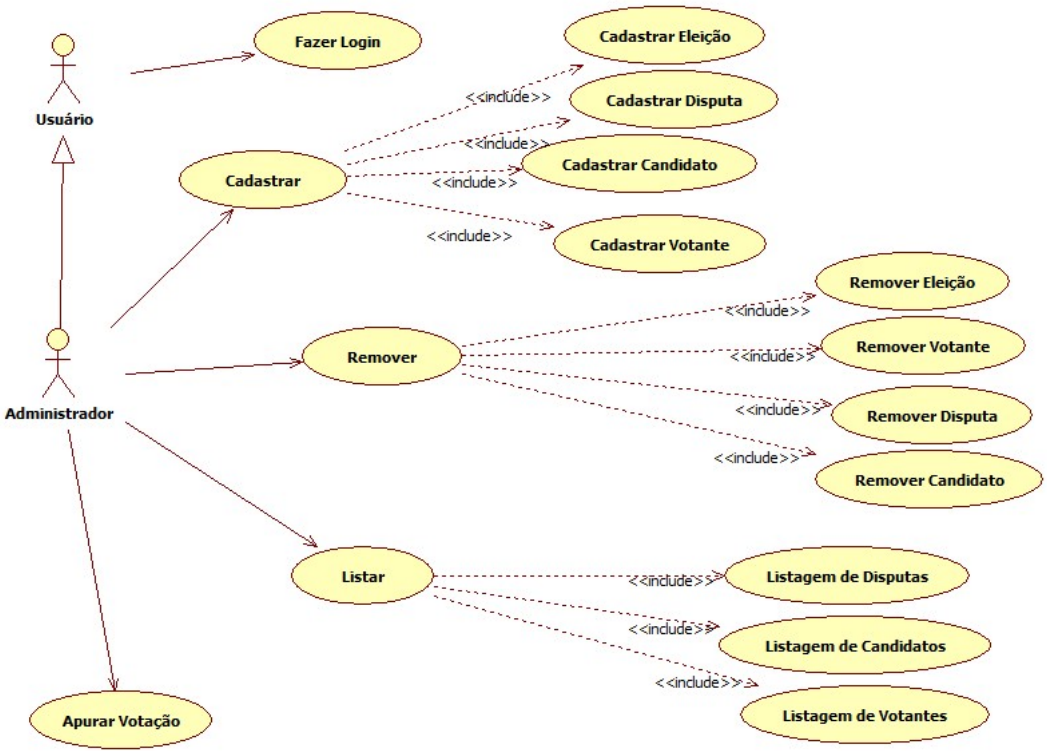

Figura 4.2: Casos de Uso Servidor

4.2.3.2 Diagrama de casos de uso do Votante

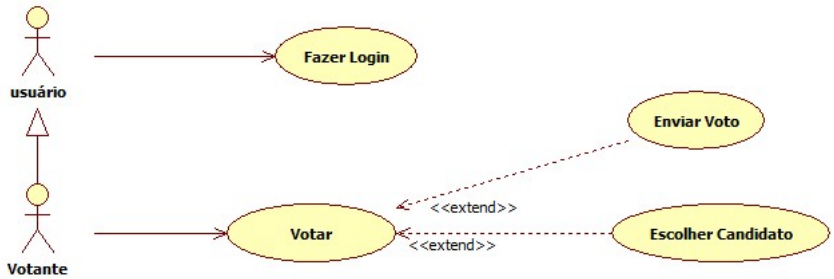

Figura 4.3: Casos de Uso Votante

4.2.4 Apresentação da Aplicação

Fase de Configuração

A aplicação se divide em 2 grandes partes. A primeira é a fase de configuração, onde o administrador da votação cadastra os votantes, os candidatos, a eleição , as disputas e todos os parâmetros necessários para a eleição como data de início, data de término e data para apresentação dos resultados, como mostrado a seguir.

A tela inicial que o administrador tem acesso após fazer login é a tela de gerência onde ele tem acesso aos recursos de listagem, cadastro, remoção de todos os recursos relacionados a votação. A seguir serão mostrados exemplos da visão do administrador do sistema.

Tela Principal

Nessa tela aparece as opções disponíveis para o perfil administrador do sistema, além da listagem das votações pendentes (ainda não iniciadas) e votações em andamento.

A seguir é apresentada a imagem da tela:

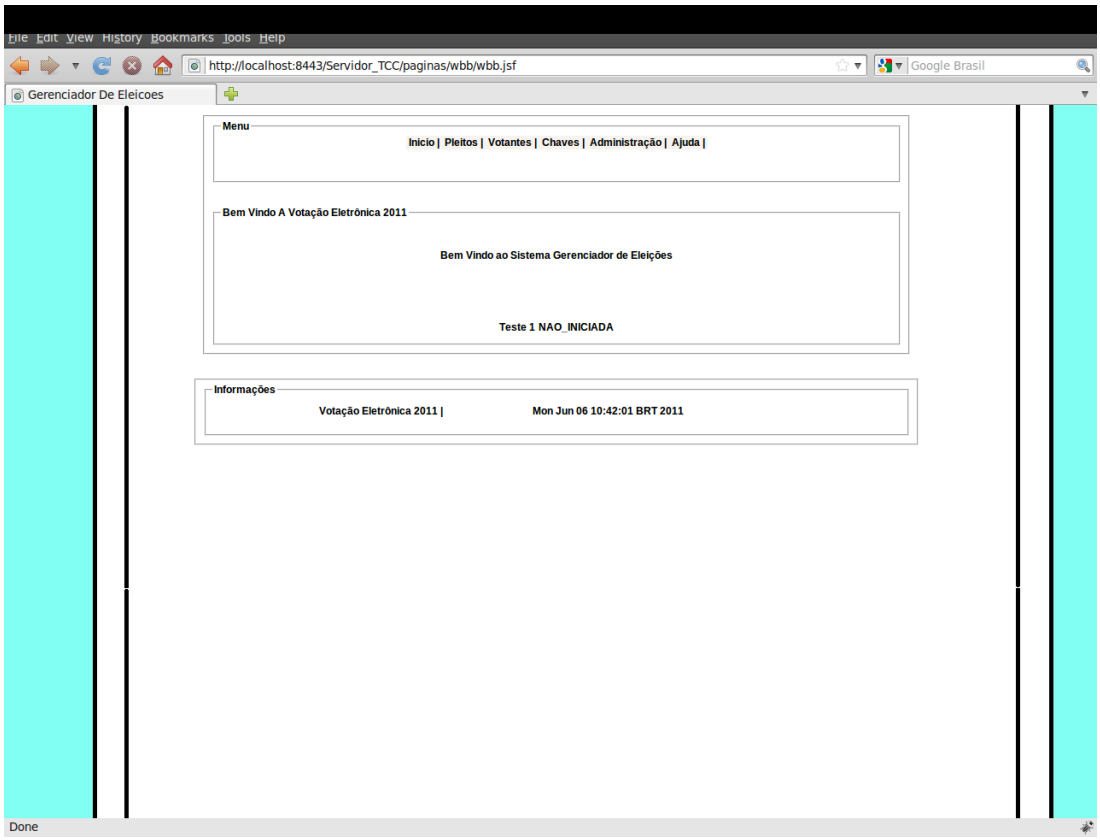

Figura 4.4: Tela Inicial

Cadastros

O administrador pode criar eleição. A eleição é a parte central do protótipo, a ela estão vinculadas as disputas que representam o objeto da votação.

A tela de cadastro de eleição possui cinco campos, (nome,data de início, data de término, tipo de autenticação, status). O campo *nome* é identificante, para não haver confusão na hora do voto, a *data de início* e *data de fim*, servem respectivamente para delimitar o tempo em que a votação poderá ser executada, o campo *tipo de autenticação* define a forma como os votantes podem logar no sistema. Para o protótipo foi definido apenas login/senha mas o ambiente está preparado para suportar outras formas de autenticação como certificado digital.

A seguir é mostrada a tela de cadastro da eleição:

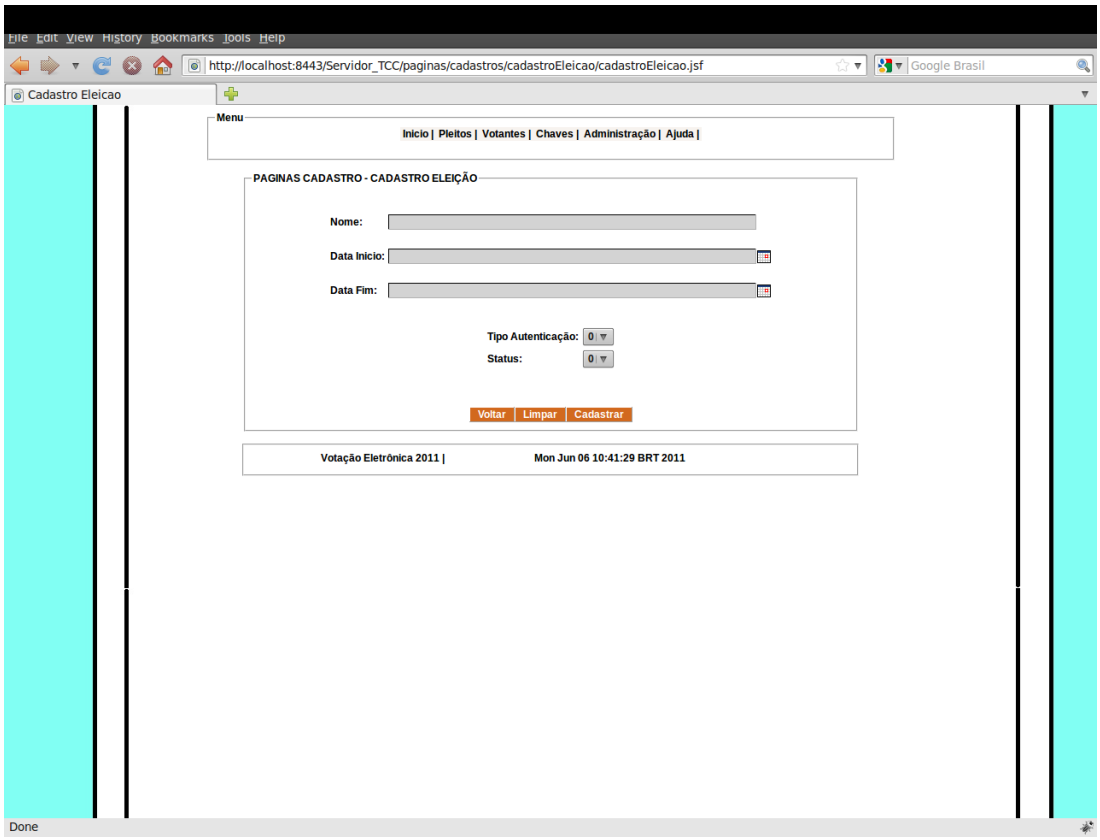

Figura 4.5: Cadastro Eleição

Os outros cadastros disponíveis para o administrador são o cadastro de disputas onde é mostrado um campo de escolha de eleição, para ser cadastrada uma disputa vinculada a eleição escolhida, o campo *nome* e *descrição* da disputa são campos de texto simples onde o administrador determina os parâmetros da disputa.

O cadastro de votante possui os campos *nome*, *data de nascimento* e *CPF*, que são os dados pessoais de cada votante, além dos campos de *login*, *senha* e o *PIN* do celular que são utilizados para logar no sistema. O *PIN* é o numero de identificação do celular, que é usado para o votante poder votar somente através do celular pré-cadastrado.

O cadastro do candidato possui os campos de nome (nome do candidato), código que é utilizado para votação por código, quando a eleição estiver definida como código um campo de foto, e os campos de combo para selecionar a disputa e a eleição

ao qual o candidato está vinculado.

Além dos cadastros, o administrador do sistema pode visualizar os dados através das listagens. A listagem de eleições ativas está disponível na tela inicial. Para as listagens de disputas e votantes há uma sessão separada no menu em que também é possível remover a entidade listada.

A seguir são mostradas 2 telas. Uma tela de referência para a visualização de cadastro e uma para visualização de como é exibida a listagem. As demais telas podem ser encontradas no apêndice.

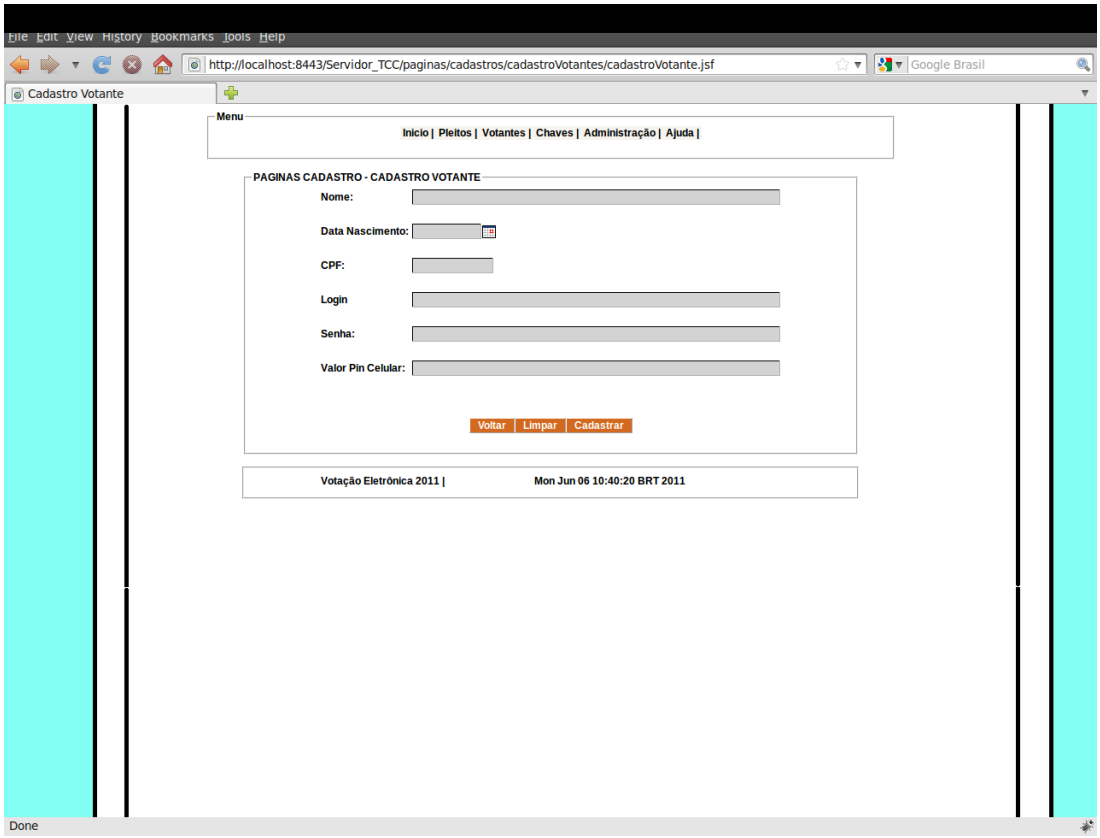

Figura 4.6: Cadastro do Votante

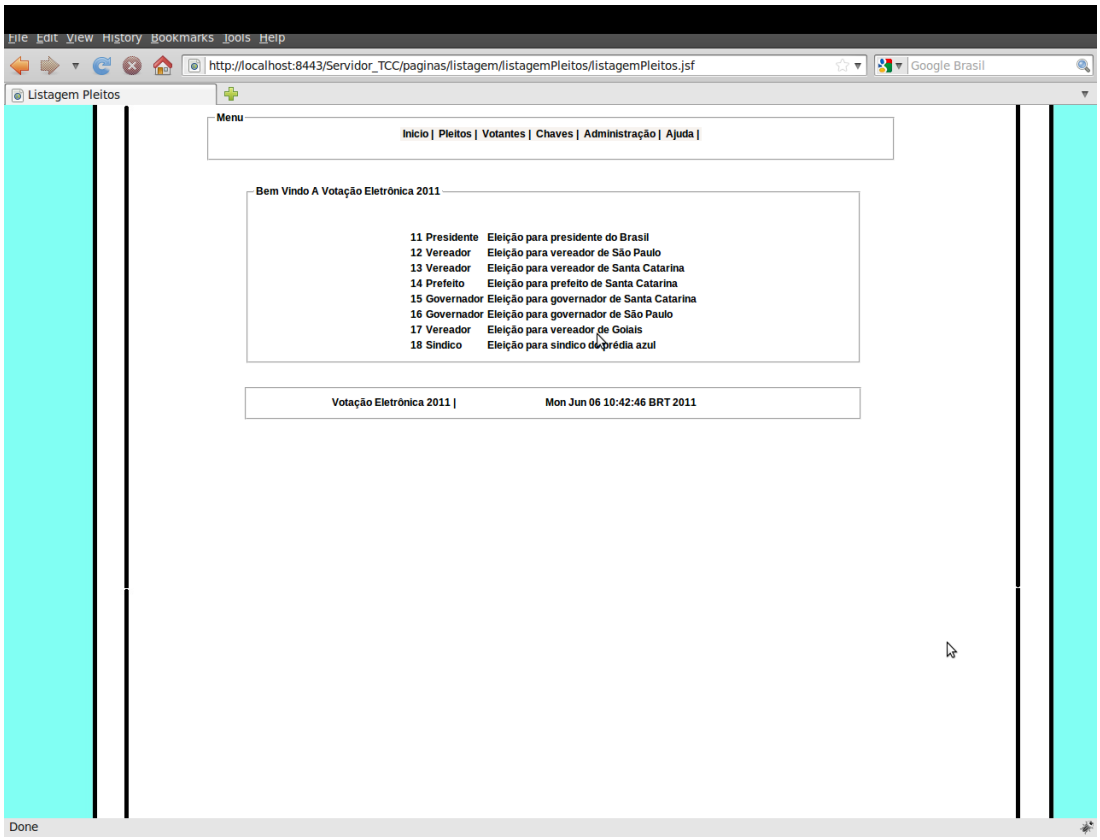

Figura 4.7: Listagem das Disputas

Fase de Votação

A segunda fase do processo de votação implementado é onde há a escolha do voto de cada um dos votantes do processo de votação. A seguir será apresentado o fluxo principal da escolha dos candidatos no cliente. A tela inicial que o votante tem acesso, ao iniciar a aplicação é a tela de *login*, na qual o votante insere seus dados de login e senha, previamente cadastrados no servidor. Se o votante conseguir logar receberá do servidor uma lista de disputas no qual ele está vinculado, juntamente com os candidatos de cada uma dessas disputas e também receberá a chave pública (ElGamal) do servidor, que será usada para cifrar o voto.

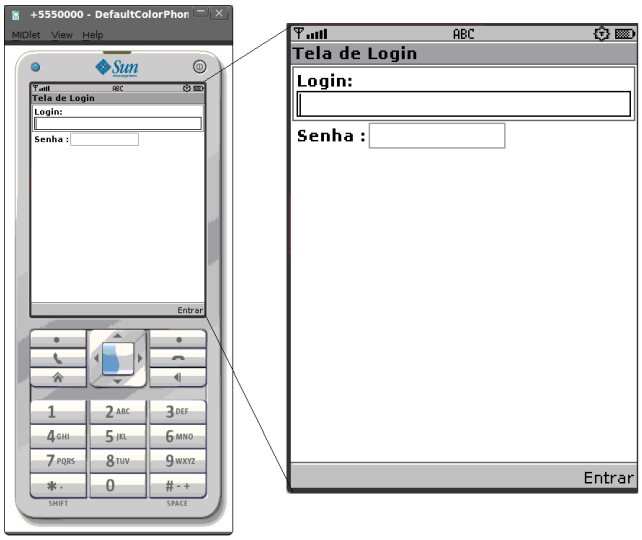

Figura 4.8: Login

Após o login, é exibido ao votante a tela de escolha do candidato, nessa tela uma disputa ao qual o votante ainda não votou é escolhida, se a eleição estiver configurada para exibição das disputas por listagem é exibida uma lista com todos os votantes da disputa escolhida.

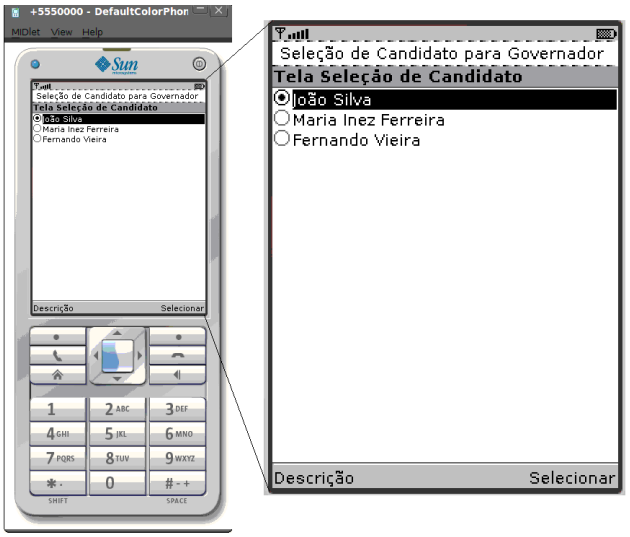

Figura 4.9: Escolha do candidato com listagem

Se for configurada a exibição por código é exibida uma tela indicando a disputa escolhida e um campo para o votante digitar o código do candidato.

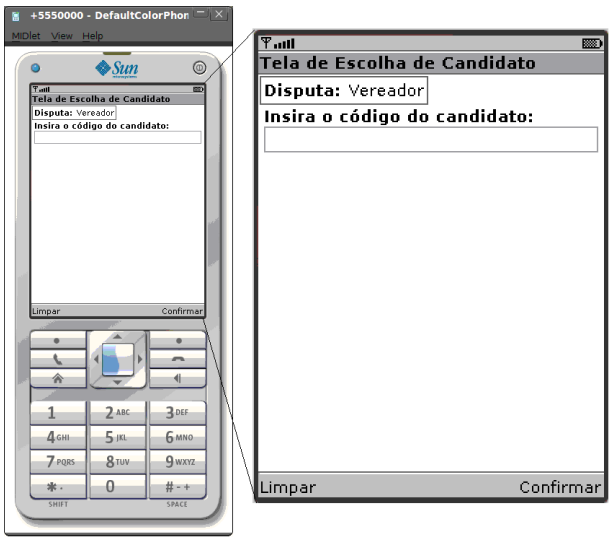

Figura 4.10: Escolha do candidato por código

Após a escolha do candidato desejado, é apresentada uma tela de confirmação com os dados do candidato escolhido a disputa ao qual ele pertence, e dois botões um com a opção de *confirmar* o voto no candidato, e outro com a opção de *cancelar*. Caso seja cancelado o voto, a aplicação volta para a tela anterior para que o votante escolha novamente o candidato da disputa.

Se o votante clicar em *confirmar* o voto uma nova disputa em que o votante não votou ainda é escolhida, e é apresentado novamente a tela de escolha do candidato. Caso essa seja a última disputa a ser votada, é exibida a tela de término da votação, se o votante logar e já tiver votado em todas as disputas essa mesma tela também é apresentada.

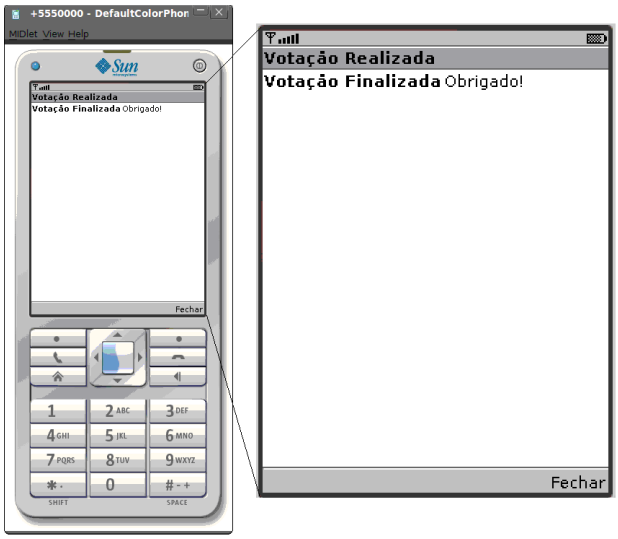

Figura 4.11: Votação Concluída

Fluxo de Votação

O fluxo de votação do sistema mostrado a seguir consiste em todas as etapas necessárias para que o votante possa realizar o seu voto. Dentre as etapas necessárias estão :

1. O votante fazer login no sistema:

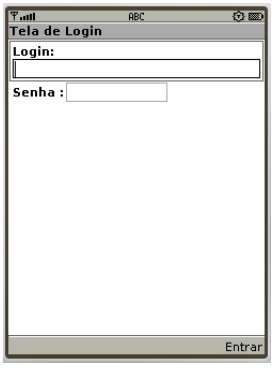

Figura 4.12: Login Celular

- Ao efetuar o login no sistema, será exibida uma tela de seleção de eleições.

2. O votante selecionar uma eleição:

Figura 4.13: Seleção de Eleição

- O votante irá selecionar uma eleição (disponível e na qual ele está cadastrado) para realizar o processo de votação.

3. O votante irá escolher um candidato:

- Nesse contexto, poderão ser exibidas 2 tipos de telas para o candidato, dependendo de como a eleição foi criada pelo administrador, se foi para votação por código ou votação por listagem.

  (a) Votação por Código:

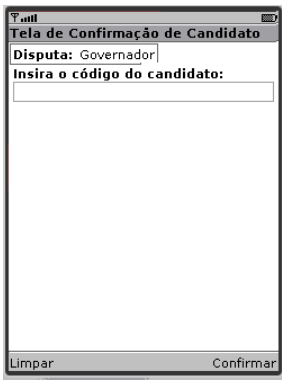

Figura 4.14: Votação por Código

  – Para cada disputa, o votante digita o código do candidato.

(b) Votação por Listagem:

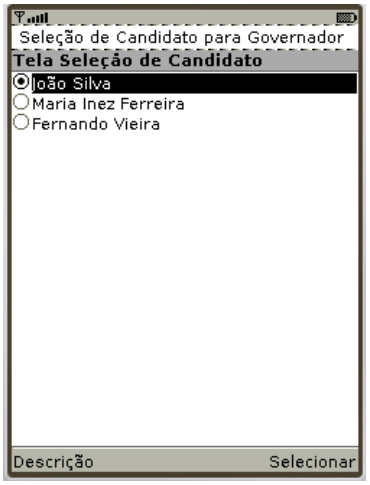

Figura 4.15: Votação por Listagem

– Para cada disputa, o votante seleciona um candidato de uma lista.

4. Confirmação de candidato:

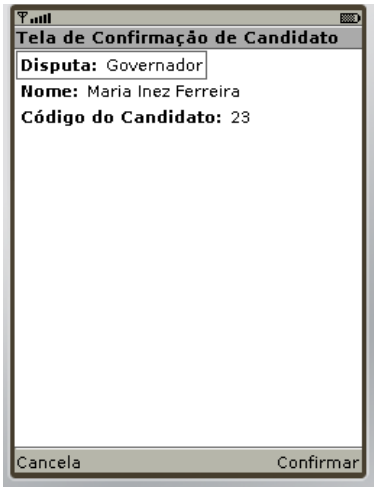

Figura 4.16: Confirmação de Candidato

- Após o votante ter escolhido seu candidato por um código ou através de uma listagem, será exibida ao votante uma tela na qual ele irá confirmar o candidato escolhido por ele para a disputa atual.

5. Votação concluída:

Figura 4.17: Votação Concluída

- Depois de o votante ter feito a escolha de um candidato por disputa, para cada uma das disputas, será exibida uma tela indicando que a votação foi finalizada.

Capítulo 5

Conclusão

Uma aplicação para votação digital em dipositivos móveis utilizando aplicações Java ME como a proposta deste trabalho tem a sua importância na exploração de um novo horizonte ainda pouco trabalhado, que é a votação eletrônica em ambientes não controlados, utilizando-se de alguns recursos de segurança necessários para uma votação confiável e segura.

Este projeto representa um passo a mais na área de votação eletrônica no Brasil, onde existe uma lacuna ainda não explorada que apresenta grandes benefícios, como mobilidade, onde o votante tem a liberdade de escolha de um local e horário em que deseja votar e o benefício econômico, pois uma votação em um ambiente controlado tem um custo muito elevado enquanto votações por meios eletrônicos tem um custo bem mais baixo.

A uso de celulares já está em crescimento há muito tempo, segundo dados da Teleco Inteligência em Comunicações [TEL 11] de abril de 2011, existem cerca de 212,6 milhões de celulares no Brasil, um número muito expressivo para uma população de cerca de 190 milhões de pessoas. A união desta tecnologia com a evolução dos protocolos para uma votação digital segura pode trazer uma série de benefícios para a sociedade.

Referências

[ALI] ALINE SOUSA DA SILVEIRA e ANTONIO CÂNDIDO FALEIROS. **CRIPTOGRAFIA DE CHAVE PÚBLICA - O PAPEL DA ARITMÉTICA EM PRECISÃO MÚLTIPLA**. Disponível em <http://www.bibl.ita.br/xiencita/Artigos/Fund05.pdf>. Acesso em: 16 de maio de 2011.

[AND 07] ANDRÉ LUIZ CARDOSO. **Uma ferramenta para obtenção on line de certificados digitais para uso de smart cards em aplicações Java ME**. Disponível em <http://www.projetos.inf.ufsc.br/arquivos_projetos/projeto_582/tcc.pdf.1>. Acesso em: 16 de maio de 2011.

[ARA 02] ARAÚJO, R. S. D. S. **Protocolos Criptográficos para Votação Digital**. Universidade Federal de Santa Catarina, 2002. Curso de pós-graduação em ciência da computação.

[ARA 08] ARAÚJO, R. S. D. S. **On Remote and Voter-Verifiable Voting**. TU Darmstadt, 2008. Ph.d. thesis.

[CRI 11] CRISTIANO FIORESI. **CONCEITOS BASICOS DAS PLATAFORMAS JAVA E J2ME**. Disponível em <http://www.devmedia.com.br/articles/viewcomp_forprint.asp?comp=6484>. Acesso em: 16 de maio de 2011.

[Cyn 10] Cynara Peixoto. **Você acha que só no Brasil tem eleição eletrônica?** Disponível em <http://www.mundotecno.info/opiniao/voce-acha-que-so-no-brasil-tem-eleicao-eletronica>. Acesso em: 20 de maio de 2011.

[DAV 09] DAVI GARCIA PEREIRA. **Assinatura Digital de Documentos Eletrônicos em Dispositivos Móveis**. Disponível em <http://www.projetos.inf.ufsc.br/arquivos_projetos/projeto_692/tcc-davigp-final.pdf>. Acesso em: 16 de maio de 2011.

[Dic 11] Dicionário Aurélio. **Dicionário Aurélio Beta**. Disponível em <http://www.dicionariodoaurelio.com/Votacao>. Acesso em: 20 de maio de 2011.

[DIE 11] DIEBOLD. **DIEBOLD**. Disponível em <http://www.diebold.com.br/>. Acesso em: 20 de maio de 2011.

[EME 05] EMERSON ALECRIM. **Criptografia**. Disponível em <http://www.infowester.com/criptografia.php>. Acesso em: 16 de maio de 2011.

[FAB 05] FABIANO CASTRO PEREIRA. **Estudo e Implementação de Redes de Comunicação Anônima e aplicação ao Sistema de Votação Digital OSTRACON**. Universidade Federal de Santa Catarina, 2005. Curso de pós-graduação em ciência da computação.

[FUN 11] FUNDAÇÃO CERTI. **CERTI**. Disponível em <http://www.certi.org.br/>. Acesso em: 20 de maio de 2011.

[IBG 11] IBGE. **Censo 2010**. Disponível em <http://www.censo2010.ibge.gov.br/>. Acesso em: 20 de maio de 2011.

[INS 11] INSTITUTO SUPERIOR TÉCNICO. **MIT Portugal**. Disponível em <http://www.ist.utl.pt>. Acesso em: 20 de maio de 2011.

[JCP 11] JCP - Java Community Process. **Community Development of Java Technology Specifications**. Disponível em <http://jcp.org/en/introduction/overview>. Acesso em: 16 de maio de 2011.

[LUÍ 06] LUÍS MIGUEL SILVA COSTA e NUNO ALEXANDRE COSTA FRETA DOS SANTOS. **MobileREVS - Votação Electrónica**. Disponível em <http://www.gsd.inesc-id.pt/m̃revs/MobileREVS_RelatorioTFC.pdf>. Acesso em: 16 de maio de 2011.

[POS 11] **Sobre o PostgreSQL**. Disponível em <http://www.postgresql.org.br/sobre>. Acesso em: 16 de maio de 2011.

[SOD 11] **Sítio do Projecto Democracia Electrónica**. Disponível em <http://e-voto.di.fc.ul.pt>. Acesso em: 20 de maio de 2011.

[Sun 11] Sun Microsystems. **Java ME**. Disponível em <http://java.sun.com/javame/technology>. Acesso em: 20 de maio de 2011.

[TEL 11] TELECO. **Teleco Inteligência em Comunicações**. Disponível em <http://www.teleco.com.br/ncel.asp>. Acesso em: 20 de maio de 2011.

[WEE 11] **Site Wikipedia. ElGamal encryption**. Disponível em <http://en.wikipedia.org/wiki/ElGamal_encryption>. Acesso em: 16 de maio de 2011.

[WES 11] **Site Wikipedia. Eclipse (software)**. Disponível em <http://en.wikipedia.org/wiki/Eclipse_%28software%29>. Acesso em: 16 de maio de 2011.

[Wik 11] Wikipédia, a enciclopédia livre. **Urna eletrônica**. Disponível em <http://pt.wikipedia.org/wiki/Urna_eletr%C3%B4nica>. Acesso em: 20 de maio de 2011.

Apêndice A

Código Fonte

A.1 Servidor

```
1
2  package br.com.dao;
3
4  import java.util.List;
5
6  import org.hibernate.Criteria;
7  import org.hibernate.Query;
8  import org.hibernate.Session;
9  import org.hibernate.SessionFactory;
10 import org.hibernate.cfg.AnnotationConfiguration;
11 import org.hibernate.criterion.Expression;
12
13 import br.com.beam.Candidato;
14 import br.com.beam.Voto;
15
16 public class CandidatoDAO {
17
18     public CandidatoDAO(){
19
20     }
21
22     private Session getSession(){
23         SessionFactory sf = new AnnotationConfiguration().configure().
               buildSessionFactory();
24         Session session = sf.openSession();
25         return session;
```

```
26      }
27
28      public void cadastrarCandidato(Candidato candidato) {
29          AnnotationConfiguration configuracao = new AnnotationConfiguration();
30          configuracao.addAnnotatedClass(Candidato.class);
31          configuracao.addAnnotatedClass(Voto.class);
32          configuracao.configure("hibernate.cfg.xml");
33          //linha abaixo apenas descomentar se a tabela ainda nao existir
34          //new SchemaExport(configuracao).create(true, true);
35          SessionFactory fabrica = configuracao.buildSessionFactory();
36          Session secao = fabrica.getCurrentSession();
37          secao.beginTransaction();
38          secao.save(candidato);
39          secao.getTransaction().commit();
40
41      }
42
43      public List<Candidato> getTodosCandidatos(){
44          try{
45          Query query = this.getSession().createQuery("from Candidato c");
46          List<Candidato> lista = (List<Candidato>) query.list();
47          return lista;
48          } catch (Exception e) {
49              e.printStackTrace();
50              return null;
51          }
52      }
53
54      public Candidato getCandidatoPorCodigo(int codigoCandidato){
55          Session session = this.getSession();
56          Query query = session.getNamedQuery("getCandidatoPorCodigo");
57          query.setParameter("codigo", codigoCandidato);
58          Candidato candidato =  (Candidato) query.uniqueResult();
59          return candidato;
60      }
61
62  }
63
64
65  package br.com.dao;
66
67
68  import java.util.List;
69
```

```
70  import org.hibernate.Criteria;
71  import org.hibernate.Query;
72  import org.hibernate.Session;
73  import org.hibernate.SessionFactory;
74  import org.hibernate.cfg.AnnotationConfiguration;
75  import org.hibernate.criterion.Expression;
76
77  import br.com.beam.Candidato;
78  import br.com.beam.Disputa;
79  import br.com.beam.Eleicao;
80  import br.com.beam.Voto;
81
82  public class DisputaDAO {
83
84
85      public DisputaDAO(){
86
87      }
88
89      private Session getSession(){
90          SessionFactory sf = new AnnotationConfiguration().configure().
                buildSessionFactory();
91          Session session = sf.openSession();
92          return session;
93      }
94
95      public void cadastraDisputa(Disputa disputa){
96          AnnotationConfiguration configuracao = new AnnotationConfiguration();
97          configuracao.addAnnotatedClass(Disputa.class);
98          configuracao.addAnnotatedClass(Voto.class);
99          configuracao.configure("hibernate.cfg.xml");
100         //linha abaixo apenas descomentar se a tabela ainda nao existir
101         //new SchemaExport(configuracao).create(true, true);
102         SessionFactory fabrica = configuracao.buildSessionFactory();
103         Session secao = fabrica.getCurrentSession();
104         secao.beginTransaction();
105         secao.save(disputa);
106         secao.getTransaction().commit();
107     }
108
109
110     @SuppressWarnings("unchecked")
111     public List<Disputa> getTodasDisputas(){
112         try{
```

```
113         Query query = this.getSession().createQuery("from Disputa d");
114         List<Disputa> lista = (List<Disputa>) query.list();
115         return lista;
116         } catch (Exception e) {
117             e.printStackTrace();
118             return null;
119         }
120     }
121 
122 
123     public Disputa getDisputaPorNome(String nomeDisputa){
124         Session session = this.getSession();
125         Query query = session.getNamedQuery("getDisputaPorNome");
126         query.setParameter("nome", nomeDisputa);
127         Disputa disputa =  (Disputa) query.uniqueResult();
128         return disputa;
129     }
130 
131     @SuppressWarnings("unchecked")
132     public List<Disputa> getDisputasPorEleicao(Eleicao id){
133         Session session = this.getSession();
134         Query query = session.getNamedQuery("getDisputasPorEleicao");
135         query.setParameter("id", id);
136         List<Disputa> listaDisputas =  (List<Disputa>) query.list();
137         return listaDisputas;
138 
139     }
140 
141 
142 
143 }
144 
145 
146 package br.com.dao;
147 
148 import java.util.List;
149 
150 import org.hibernate.Criteria;
151 import org.hibernate.Query;
152 import org.hibernate.Session;
153 import org.hibernate.SessionFactory;
154 import org.hibernate.cfg.AnnotationConfiguration;
155 import org.hibernate.criterion.Expression;
156 import org.hibernate.tool.hbm2ddl.SchemaExport;
```

```
157
158 import br.com.beam.Candidato;
159 import br.com.beam.Disputa;
160 import br.com.beam.Eleicao;
161 import br.com.beam.StatusEleicao;
162 import br.com.beam.Votante;
163 import br.com.beam.VotanteEleicao;
164
165 public class EleicaoDAO {
166
167     public EleicaoDAO(){
168
169     }
170
171     private Session getSession(){
172         SessionFactory sf = new AnnotationConfiguration().configure().
                buildSessionFactory();
173         Session session = sf.openSession();
174         return session;
175     }
176
177     public void cadastrarEleicao(Eleicao eleicao) {
178
179         AnnotationConfiguration configuracao = new AnnotationConfiguration();
180         configuracao.addAnnotatedClass(Eleicao.class);
181         configuracao.addAnnotatedClass(Votante.class);
182         configuracao.addAnnotatedClass(Disputa.class);
183         configuracao.addAnnotatedClass(Candidato.class);
184         configuracao.configure("hibernate.cfg.xml");
185         //linha abaixo apenas descomentar se a tabela ainda nao existir
186         if (this.getTodasEleicoes()==null){
187         new SchemaExport(configuracao).create(true, true);}
188         SessionFactory fabrica = configuracao.buildSessionFactory();
189         Session secao = fabrica.getCurrentSession();
190         secao.beginTransaction();
191         secao.save(eleicao);
192         secao.getTransaction().commit();
193     }
194
195     @SuppressWarnings("unchecked")
196     public List<Eleicao> getTodasEleicoes(){
197         try{
198         Query query = this.getSession().createQuery("from Eleicao e");
199         List<Eleicao> lista = (List<Eleicao>) query.list();
```

```
200         return lista;
201         } catch (Exception e) {
202             return null;
203         }
204     }
205 
206     @SuppressWarnings("unchecked")
207     public List<Eleicao> getEleicaoDisponivelDoVotante(List<Long> listaPassada){
208         try{
209         String pedacoQuery = "";
210         String andIdDif = " and e.id<> ";
211         if (listaPassada.size()>0){
212             pedacoQuery = " where e.id<> "+listaPassada.get(0).toString();
213             for (int i=1;i<listaPassada.size();i++){
214                 pedacoQuery = pedacoQuery + andIdDif + listaPassada.get(i).toString()
                        ;
215             }
216         }
217         Query query = this.getSession().createQuery("from Eleicao e "+pedacoQuery);
218         List<Eleicao> lista = (List<Eleicao>) query.list();
219         return lista;
220         } catch (Exception e) {
221             return null;
222         }
223     }
224 
225     @SuppressWarnings("unchecked")
226     public List<Eleicao> getEleicaoPorStatus(StatusEleicao status){
227         Session session = this.getSession();
228         Query query = session.getNamedQuery("getEleicaoPorStatus");
229         query.setParameter("status", status);
230         List<Eleicao> listaEleicao =  (List<Eleicao>) query.list();
231         return listaEleicao;
232     }
233 
234 
235     public Eleicao getEleicaoPorNome(String nomeEleicao){
236         Session session = this.getSession();
237         Query query = session.getNamedQuery("getEleicaoPorNome");
238         query.setParameter("nome", nomeEleicao);
239         Eleicao eleicao =  (Eleicao) query.uniqueResult();
240         return eleicao;
241     }
242 
```

```
243
244     public Eleicao getEleicaoPorId(Long id){
245         Session session = this.getSession();
246         Query query = session.getNamedQuery("getEleicaoPorId");
247         query.setParameter("id", id);
248         Eleicao eleicao =  (Eleicao) query.uniqueResult();
249         return eleicao;
250     }
251
252 }
253
254
255 package br.com.dao;
256
257 import org.hibernate.Session;
258 import org.hibernate.SessionFactory;
259 import org.hibernate.cfg.AnnotationConfiguration;
260
261 import br.com.beam.PinEleicao;
262
263 public class PinEleicaoDAO {
264
265
266     private Session getSession(){
267         SessionFactory sf = new AnnotationConfiguration().configure().
                buildSessionFactory();
268         Session session = sf.openSession();
269         return session;
270     }
271
272     public void cadastrarPinEleicao(PinEleicao pinEleicao) {
273         AnnotationConfiguration configuracao = new AnnotationConfiguration();
274         configuracao.addAnnotatedClass(PinEleicao.class);
275         configuracao.configure("hibernate.cfg.xml");
276         //linha abaixo apenas descomentar se a tabela ainda nao existir
277         //new SchemaExport(configuracao).create(true, true);
278         SessionFactory fabrica = configuracao.buildSessionFactory();
279         Session secao = fabrica.getCurrentSession();
280         secao.beginTransaction();
281
282         secao.save(pinEleicao);
283
284         secao.getTransaction().commit();
285
```

```
286         }
287     }
288
289
290     package br.com.dao;
291
292     import java.util.List;
293
294     import org.hibernate.Criteria;
295     import org.hibernate.Query;
296     import org.hibernate.Session;
297     import org.hibernate.SessionFactory;
298     import org.hibernate.cfg.AnnotationConfiguration;
299     import org.hibernate.criterion.Expression;
300
301     import br.com.beam.Eleicao;
302     import br.com.beam.Login\_Senha;
303     import br.com.beam.Votante;
304
305     public class VotanteDAO {
306
307         public VotanteDAO(){
308
309         }
310
311         private Session getSession(){
312             SessionFactory sf = new AnnotationConfiguration().configure().
                    buildSessionFactory();
313             Session session = sf.openSession();
314             return session;
315         }
316
317         public void cadastrarVotante(Votante votante) {
318             AnnotationConfiguration configuracao = new AnnotationConfiguration();
319             configuracao.addAnnotatedClass(Votante.class);
320             configuracao.addAnnotatedClass(Login\_Senha.class);
321             configuracao.configure("hibernate.cfg.xml");
322             //linha abaixo apenas descomentar se a tabela ainda nao existir
323             //new SchemaExport(configuracao).create(true, true);
324             SessionFactory fabrica = configuracao.buildSessionFactory();
325             Session secao = fabrica.getCurrentSession();
326             secao.beginTransaction();
327
328             Login\_Senha loginSenha = votante.getFk\_login\_Senha();
```

```
329             loginSenha.setFk\_votante(votante);
330             votante.setFk\_login\_Senha(loginSenha);
331             secao.save(votante);
332             secao.save(loginSenha);
333 
334             secao.getTransaction().commit();
335 
336         }
337 
338         public List<Votante> getTodosVotantes(){
339             try{
340             Query query = this.getSession().createQuery("from Votante v");
341             List<Votante> lista = (List<Votante>) query.list();
342             return lista;
343             } catch (Exception e) {
344                 e.printStackTrace();
345                 return null;
346             }
347         }
348 
349 
350         public Votante getVotantePorCodigo(int codigoVotante){
351             Criteria consulta = this.getSession().createCriteria(Votante.class);
352             consulta.add( Expression.like("codigo", codigoVotante) );
353             Votante votante =  (Votante) consulta.uniqueResult();
354             return votante;
355         }
356 
357         public Votante getVotantePorIdentificacao(String identificacao){
358             Session session = this.getSession();
359             Query query = session.getNamedQuery("getVotantePorIdentificacao");
360             query.setParameter("identificacao", identificacao);
361             Votante votante =  (Votante) query.uniqueResult();
362             return votante;
363         }
364 
365     }
366 
367 
368     package br.com.dao;
369 
370     import java.util.List;
371 
372     import org.hibernate.Criteria;
```

```
373 import org.hibernate.Query;
374 import org.hibernate.Session;
375 import org.hibernate.SessionFactory;
376 import org.hibernate.cfg.AnnotationConfiguration;
377 import org.hibernate.criterion.Expression;
378
379 import br.com.beam.Eleicao;
380 import br.com.beam.Login\_Senha;
381 import br.com.beam.StatusEleicao;
382 import br.com.beam.Votante;
383 import br.com.beam.VotanteEleicao;
384
385 public class VotanteEleicaoDAO {
386
387     public VotanteEleicaoDAO() {
388
389     }
390
391     private Session getSession() {
392         SessionFactory sf = new AnnotationConfiguration().configure()
393                 .buildSessionFactory();
394         Session session = sf.openSession();
395         return session;
396     }
397
398     public void cadastrarVotanteEleicao(VotanteEleicao votanteEleicao) {
399         AnnotationConfiguration configuracao = new AnnotationConfiguration();
400         SessionFactory fabrica = configuracao.buildSessionFactory();
401         Session secao = fabrica.getCurrentSession();
402         secao.beginTransaction();
403         try {
404             configuracao.addAnnotatedClass(VotanteEleicao.class);
405             configuracao.configure("hibernate.cfg.xml");
406             // linha abaixo apenas descomentar se a tabela ainda nao existir
407             // new SchemaExport(configuracao).create(true, true);
408             secao.save(votanteEleicao);
409             secao.getTransaction().commit();
410         } catch (Exception e) {
411             // TODO: handle exception
412         } finally{
413             secao.close();
414         }
415
416
```

```
417
418
419         }
420
421
422         @SuppressWarnings("unchecked")
423         public List<VotanteEleicao> getEleicaoDoVotantePorIdDoVotante(Votante id){
424             Session session = this.getSession();
425             Query query = session.getNamedQuery("getVotanteEleicaoPorIdDoVotante");
426             query.setParameter("id", id);
427             List<VotanteEleicao> listaEleicao =  (List<VotanteEleicao>) query.list();
428             return listaEleicao;
429         }
430
431
432
433     }
434
435
436     package br.com.dto;
437
438     import br.com.beam.Eleicao;
439     import br.com.beam.Votante;
440     import br.com.beam.VotanteEleicao;
441
442     public class VotanteEleicaoDTO {
443
444         private Votante votante;
445         private Eleicao eleicao;
446
447
448         public VotanteEleicao preencherVotanteEleicao(VotanteEleicao votanteEleicao) {
449             votanteEleicao.setVotante(getVotante());
450             //votanteEleicao.setEleicao(getEleicao());
451             return votanteEleicao;
452
453         }
454
455         public void preencherDTO(VotanteEleicao votanteEleicao) {
456             this.setEleicao(votanteEleicao.getEleicao());
457             //this.setVotante(votanteEleicao.getVotante());
458
459         }
460
```

```
461     public void setVotante(Votante votante) {
462         this.votante = votante;
463     }
464 
465     public Votante getVotante() {
466         return votante;
467     }
468 
469     public void setEleicao(Eleicao eleicao) {
470         this.eleicao = eleicao;
471     }
472 
473     public Eleicao getEleicao() {
474         return eleicao;
475     }
476 
477 }
478 
479 
480 package br.com.war;
481 
482 import java.awt.image.BufferedImage;
483 import java.io.BufferedInputStream;
484 import java.io.ByteArrayInputStream;
485 import java.io.IOException;
486 import java.io.OutputStream;
487 import java.util.ArrayList;
488 import java.util.List;
489 
490 import javax.faces.event.PhaseId;
491 import javax.faces.event.ValueChangeEvent;
492 import javax.faces.model.SelectItem;
493 import javax.imageio.ImageIO;
494 
495 import org.apache.myfaces.custom.fileupload.UploadedFile;
496 
497 import br.com.beam.Candidato;
498 import br.com.beam.Disputa;
499 import br.com.beam.Eleicao;
500 import br.com.dao.CandidatoDAO;
501 import br.com.dao.DisputaDAO;
502 import br.com.dao.EleicaoDAO;
503 import br.com.dto.CandidatoDTO;
504 
```

```
505 public class CandidatoBB {
506
507     private String nome;
508     private int codigo;
509     private CandidatoDAO candidatoDAO;
510     private Candidato candidato;
511     private List<SelectItem> listaDisputaExibir;
512     private String disputaSelecionada;
513     private DisputaDAO disputaDAO;
514     private List<Disputa> listaDisputas;
515     private transient UploadedFile uploadFoto;
516     private CandidatoDTO candidatoDTO;
517     private String eleicaoSelecionada;
518     private List<Eleicao> listaEleicoes;
519     private EleicaoDAO eleicaoDAO;
520     private String codigoString;
521
522     public CandidatoBB() {
523         candidatoDAO = new CandidatoDAO();
524         candidato = new Candidato();
525         disputaDAO = new DisputaDAO();
526         candidatoDTO = new CandidatoDTO();
527         eleicaoDAO = new EleicaoDAO();
528         this.getListaEleicaoSelectItem();
529         this.getListaDisputaSelectItem();
530     }
531
532     public String cadastrarCandidato() {
533         Candidato candidatoBanco = candidatoDAO.getCandidatoPorCodigo(this
534                 .getCodigo());
535         String retorno = "";
536         if (candidatoBanco == null) {
537             candidato.setNome(this.getNome());
538             candidato.setCodigo(Integer.parseInt(this.getCodigoString()));
539             candidato.setDisputa(disputaDAO
540                     .getDisputaPorNome(this.disputaSelecionada));
541             candidatoDAO.cadastrarCandidato(candidato);
542             retorno = "candidatoCadastrado";
543         } else {
544             retorno = "erroCodigoCandidato";
545         }
546         return retorno;
547     }
548
```

```
549     public String voltarPaginaInicial() {
550         return "voltar";
551     }
552 
553     public void paintImage(OutputStream os, Object data) throws IOException {
554         BufferedImage img = ImageIO.read(new BufferedInputStream(
555                 new ByteArrayInputStream(this.candidatoDTO.getFoto())));
556         ImageIO.write(img, "jpeg", os);
557     }
558 
559     public List<SelectItem> getListaDisputaSelectItem() {
560         List<SelectItem> listaSelectItem = new ArrayList<SelectItem>();
561         Eleicao eleicao;
562         if (this.getEleicaoSelecionada() != null){
563             eleicao = eleicaoDAO.getEleicaoPorNome(this.getEleicaoSelecionada());
564         }
565         else {
566             eleicao = eleicaoDAO.getTodasEleicoes().get(0);
567         }
568         listaDisputas = disputaDAO.getDisputasPorEleicao(eleicao);
569         for (int i = 0; i < listaDisputas.size(); i++) {
570             listaSelectItem.add(new SelectItem(listaDisputas.get(i).getNome()));
571         }
572         return listaSelectItem;
573     }
574 
575     public List<SelectItem> getListaEleicaoSelectItem(){
576         List<SelectItem> listaSelectItem = new ArrayList<SelectItem>();
577         listaEleicoes = eleicaoDAO.getTodasEleicoes();
578         for (int i=0;i<listaEleicoes.size();i++){
579             listaSelectItem.add(new SelectItem(listaEleicoes.get(i).getNome()));
580         }
581         return listaSelectItem;
582     }
583 
584 
585     public void selecionarEleicao(ValueChangeEvent evt) {
586         if (evt.getPhaseId().equals(PhaseId.ANY\_PHASE)) {
587             evt.setPhaseId(PhaseId.UPDATE\_MODEL\_VALUES);
588             evt.queue();
589         }
590         if (evt.getPhaseId().equals(PhaseId.UPDATE\_MODEL\_VALUES)) {
591             if (this.getEleicaoSelecionada() != null) {
592                 this.setEleicaoSelecionada(this.eleicaoSelecionada);
```

```
593                 }
594             }
595         }
596
597
598
599         public String getNome() {
600             return nome;
601         }
602
603         public void setNome(String nome) {
604             this.nome = nome;
605         }
606
607         public int getCodigo() {
608             return codigo;
609         }
610
611         public void setCodigo(int codigo) {
612             this.codigo = codigo;
613         }
614
615         public void setListaDisputaExibir(List<SelectItem> listaDisputaExibir) {
616             this.listaDisputaExibir = listaDisputaExibir;
617         }
618
619         public List<SelectItem> getListaDisputaExibir() {
620             return listaDisputaExibir;
621         }
622
623         public void setDisputaSelecionada(String disputaSelecionada) {
624             this.disputaSelecionada = disputaSelecionada;
625         }
626
627         public String getDisputaSelecionada() {
628             return disputaSelecionada;
629         }
630
631         public void setUploadFoto(UploadedFile uploadFoto) throws IOException {
632             this.candidatoDTO.setFoto(uploadFoto.getBytes());
633             this.uploadFoto = uploadFoto;
634         }
635
636         public UploadedFile getUploadFoto() {
```

```
637             return this.uploadFoto;
638         }
639 
640         public void setEleicaoSelecionada(String eleicaoSelecionada) {
641             this.eleicaoSelecionada = eleicaoSelecionada;
642         }
643 
644         public String getEleicaoSelecionada() {
645             return eleicaoSelecionada;
646         }
647 
648         public void setCodigoString(String codigoString) {
649             this.codigoString = codigoString;
650         }
651 
652         public String getCodigoString() {
653             return codigoString;
654         }
655 
656 }
657 
658 
659 package br.com.war;
660 
661 import java.util.ArrayList;
662 import java.util.List;
663 
664 import javax.faces.context.FacesContext;
665 import javax.faces.model.SelectItem;
666 import javax.servlet.http.HttpServletRequest;
667 import javax.servlet.http.HttpSession;
668 
669 import br.com.beam.Eleicao;
670 import br.com.beam.Votante;
671 import br.com.beam.VotanteEleicao;
672 import br.com.dao.EleicaoDAO;
673 import br.com.dao.VotanteDAO;
674 import br.com.dao.VotanteEleicaoDAO;
675 import br.com.dto.VotanteEleicaoDTO;
676 
677 public class VotanteEleicaoBB {
678 
679     private Eleicao eleicao;
680     private Votante votante;
```

```
681     private VotanteEleicao votanteEleicao;
682     private VotanteEleicaoDAO votanteEleicaoDAO;
683     private EleicaoDAO eleicaoDAO;
684     private String eleicaoSelecionada;
685     private VotanteEleicaoDTO votanteEleicaoDTO;
686     private String valorCPF;
687     private VotanteDAO votanteDAO;
688     private boolean votanteExiste = false;
689
690     public VotanteEleicaoBB() {
691         votanteEleicaoDAO = new VotanteEleicaoDAO();
692         eleicaoDAO = new EleicaoDAO();
693         votanteEleicao = new VotanteEleicao();
694         votanteDAO = new VotanteDAO();
695         setVotanteEleicaoDTO(new VotanteEleicaoDTO());
696
697     }
698
699     public String cadastrarVotanteEleicao() {
700         votanteEleicao.setEleicao(eleicaoDAO.getEleicaoPorNome(this.
                getEleicaoSelecionada()));
701         votanteEleicao.setVotante(this.votante);
702         votanteEleicaoDAO.cadastrarVotanteEleicao(votanteEleicao);
703         return "votanteEleicaoCadastrado";
704     }
705
706     public List<Eleicao> getListaEleicaoDoVotanteEleicaoEmQueOVotanteNaoEstaContido()
            {
707         if (votante!=null){
708             Long id = votante.getId();
709             setVotanteExiste(true);
710         }
711
712         List<VotanteEleicao> listaVotanteEleicao = votanteEleicaoDAO.
                getEleicaoDoVotantePorIdDoVotante(votante);
713         List<Long> listaDeIdDasEleicoesQueOVotanteEstaCadastrado = new ArrayList<Long
                >();
714         if (listaVotanteEleicao!=null){
715             for (int i=0;i<listaVotanteEleicao.size();i++){
716                 listaDeIdDasEleicoesQueOVotanteEstaCadastrado.add(i,
                        listaVotanteEleicao.get(i).getEleicao().getId());
717             }
718         }
719         else {
```

```
720
721             }
722             return eleicaoDAO.getEleicaoDisponivelDoVotante(
                    listaDeIdDasEleicoesQueOVotanteEstaCadastrado);
723         }
724
725         public String buscarVotantePorCPF(){
726             this.setVotante(votanteDAO.getVotantePorIdentificacao(this.getValorCPF()));
727             this.getListaEleicaoSelectItem();
728             return "buscarVotantePorCPFSucesso";
729         }
730
731         public List<SelectItem> getListaEleicaoSelectItem(){
732             List<SelectItem> listaSelectItem = new ArrayList<SelectItem>();
733             List<Eleicao> listaEleicoesDisponiveis = this.
                    getListaEleicaoDoVotanteEleicaoEmQueOVotanteNaoEstaContido();
734             for (int i=0;i<listaEleicoesDisponiveis.size();i++){
735                 listaSelectItem.add(new SelectItem(listaEleicoesDisponiveis.get(i).
                        getNome()));
736             }
737             return listaSelectItem;
738         }
739
740
741
742         public void setEleicaoSelecionada(String eleicaoSelecionada) {
743             this.eleicaoSelecionada = eleicaoSelecionada;
744         }
745
746         public String getEleicaoSelecionada() {
747             return eleicaoSelecionada;
748         }
749
750         public void setEleicao(Eleicao eleicao) {
751             this.eleicao = eleicao;
752         }
753
754         public Eleicao getEleicao() {
755             return eleicao;
756         }
757
758         public void setVotante(Votante votante) {
759             this.votante = votante;
760         }
```

```
761
762     public Votante getVotante() {
763         return votante;
764     }
765
766     public void setVotanteEleicaoDTO(VotanteEleicaoDTO votanteEleicaoDTO) {
767         this.votanteEleicaoDTO = votanteEleicaoDTO;
768     }
769
770     public VotanteEleicaoDTO getVotanteEleicaoDTO() {
771         return votanteEleicaoDTO;
772     }
773
774     public void setValorCPF(String valorCPF) {
775         this.valorCPF = valorCPF;
776     }
777
778     public String getValorCPF() {
779         return valorCPF;
780     }
781
782     public void setVotanteExiste(boolean votanteExiste) {
783         this.votanteExiste = votanteExiste;
784     }
785
786     public boolean isVotanteExiste() {
787         return votanteExiste;
788     }
789
790 }
791
792
793 package br.com.war;
794
795 import java.util.ArrayList;
796 import java.util.Date;
797 import java.util.List;
798
799 import javax.faces.context.FacesContext;
800 import javax.faces.model.SelectItem;
801 import javax.servlet.http.HttpServletRequest;
802 import javax.servlet.http.HttpSession;
803
804 import br.com.beam.Eleicao;
```

```
805 import br.com.beam.login\_Senha;
806 import br.com.beam.StatusVotante;
807 import br.com.beam.TipoIdentificacao;
808 import br.com.beam.Votante;
809 import br.com.dao.EleicaoDAO;
810 import br.com.dao.VotanteDAO;
811 import br.com.dto.VotanteEleicaoDTO;
812
813 public class VotanteBB {
814
815     private String nome;
816     private Date dataNascimento;
817     private String identificacao;
818     private TipoIdentificacao tipoIdentificacao;
819     private StatusVotante status;
820     private String senha;
821     private String valorPin;
822     private VotanteDAO votanteDAO;
823     private Votante votante;
824     private String tipoIdentSelecionada;
825     private String statusSelecionado;
826     private EleicaoDAO eleicaoDAO;
827     private String login;
828     private VotanteEleicaoDTO votanteEleicaoDTO;
829
830     public VotanteBB() {
831         votanteDAO = new VotanteDAO();
832         votante = new Votante();
833         votanteEleicaoDTO = new VotanteEleicaoDTO();
834     }
835
836     public String cadastrarVotante() {
837         login\_Senha loginSenha = new login\_Senha();
838         loginSenha.setLogin(this.getLogin());
839         loginSenha.setSenha(this.getSenha());
840         votante.setNome(this.getNome());
841         votante.setDataNascimento(this.getDataNascimento());
842         votante.setIdentificacao(this.getIdentificacao());
843         votante.setStatus(status.NAO\_VOTOU);
844         //this.selecionaTipoIdentificacao();
845         votante.setTipoIdentificacao(tipoIdentificacao.CPF);
846         votante.setFk\_login\_Senha(loginSenha);
847         votante.setValorPin(this.getValorPin());
848         votanteDAO.cadastrarVotante(votante);
```

```
849
850         FacesContext context = FacesContext.getCurrentInstance();
851         HttpServletRequest myRequest = (HttpServletRequest)context.getExternalContext
                ().getRequest();
852         HttpSession mySession = myRequest.getSession();
853         String atributo = (String) mySession.getAttribute("identificacaoVotante");
854         String parametro = myRequest.getParameter("identificacaoVotante");
855         String identificacao = FacesContext.getCurrentInstance().getExternalContext()
                .getRequestParameterMap().get("identificacaoVotante");
856         String temp[] = FacesContext.getCurrentInstance().getExternalContext().
                getRequestParameterValuesMap().get("identificacaoVotante");
857         for (int i=0;i<temp.length;i++){
858         }
859         votanteEleicaoDTO.setVotante(votante);
860         return "votanteCadastrado";
861     }
862
863     public Eleicao retornaPrimeiraEleicaoCriada() {
864         eleicaoDAO = new EleicaoDAO();
865         return eleicaoDAO.getTodasEleicoes().get(0);
866     }
867
868     public void selecionaStatus() {
869         if (statusSelecionado.equalsIgnoreCase("0"))
870             votante.setStatus(StatusVotante.NAO\_AUTORIZADO);
871         else if (statusSelecionado.equalsIgnoreCase("1"))
872             votante.setStatus(StatusVotante.NAO\_VOTOU);
873         else if (statusSelecionado.equalsIgnoreCase("2"))
874             votante.setStatus(StatusVotante.JA\_VOTOU);
875     }
876
877     public void selecionaTipoIdentificacao() {
878         if (tipoIdentSelecionada.equalsIgnoreCase("0"))
879             votante.setTipoIdentificacao(TipoIdentificacao.RG);
880         else if (tipoIdentSelecionada.equalsIgnoreCase("1"))
881             votante.setTipoIdentificacao(TipoIdentificacao.CPF);
882         else
883             votante.setTipoIdentificacao(TipoIdentificacao.NUMERO\_AUTORIDADE);
884     }
885
886     public List<String> listaTipoIdentificacao() {
887         List<String> listaTipoIdent = new ArrayList<String>();
888         for (int i = 0; i < 3; i++) {
889             listaTipoIdent.add("" + i);
```

```
890         }
891         return listaTipoIdent;
892     }
893 
894     public List<SelectItem> getListaTipoIdentificacao() {
895         List<SelectItem> listaSelectItem = new ArrayList<SelectItem>();
896         for (int i = 0; i < this.listaTipoIdentificacao().size(); i++) {
897             listaSelectItem
898                     .add(new SelectItem(listaTipoIdentificacao().get(i)));
899         }
900 
901         return listaSelectItem;
902     }
903 
904     public List<String> listaStatusString() {
905         List<String> lista = new ArrayList<String>();
906         for (int i = 0; i < 3; i++) {
907             lista.add("" + i);
908         }
909         return lista;
910     }
911 
912     public List<SelectItem> getListaStatus() {
913         List<SelectItem> listaSelectItem = new ArrayList<SelectItem>();
914         for (int i = 0; i < this.listaStatusString().size(); i++) {
915             listaSelectItem.add(new SelectItem(listaStatusString().get(i)));
916         }
917 
918         return listaSelectItem;
919     }
920 
921     public List<Votante> getListaVotantes() {
922         VotanteDAO dao = new VotanteDAO();
923         List<Votante> lista = dao.getTodosVotantes();
924         return lista;
925     }
926 
927 
928 
929     public String getNome() {
930         return nome;
931     }
932 
933     public void setNome(String nome) {
```

```
934             this.nome = nome;
935         }
936 
937         public Date getDataNascimento() {
938             return dataNascimento;
939         }
940 
941         public void setDataNascimento(Date dataNascimento) {
942             this.dataNascimento = dataNascimento;
943         }
944 
945         public String getIdentificacao() {
946             return identificacao;
947         }
948 
949         public void setIdentificacao(String identificacao) {
950             this.identificacao = identificacao;
951         }
952 
953         public TipoIdentificacao getTipoIdentificacao() {
954             return tipoIdentificacao;
955         }
956 
957         public void setTipoIdentificacao(TipoIdentificacao tipoIdentificacao) {
958             this.tipoIdentificacao = tipoIdentificacao;
959         }
960 
961         public StatusVotante getStatus() {
962             return status;
963         }
964 
965         public void setStatus(StatusVotante status) {
966             this.status = status;
967         }
968 
969         public String getSenha() {
970             return senha;
971         }
972 
973         public void setSenha(String senha) {
974             this.senha = senha;
975         }
976 
977         public String getValorPin() {
```

```
978             return valorPin;
979         }
980 
981         public void setValorPin(String valorPin) {
982             this.valorPin = valorPin;
983         }
984 
985         public void setTipoIdentSelecionada(String tipoIdentSelecionada) {
986             this.tipoIdentSelecionada = tipoIdentSelecionada;
987         }
988 
989         public String getTipoIdentSelecionada() {
990             return tipoIdentSelecionada;
991         }
992 
993         public void setStatusSelecionado(String statusSelecionado) {
994             this.statusSelecionado = statusSelecionado;
995         }
996 
997         public String getStatusSelecionado() {
998             return statusSelecionado;
999         }
1000 
1001        public void setLogin(String login) {
1002            this.login = login;
1003        }
1004 
1005        public String getLogin() {
1006            return login;
1007        }
1008 
1009        public void setVotanteEleicaoDTO(VotanteEleicaoDTO votanteEleicaoDTO) {
1010            this.votanteEleicaoDTO = votanteEleicaoDTO;
1011        }
1012 
1013        public VotanteEleicaoDTO getVotanteEleicaoDTO() {
1014            return votanteEleicaoDTO;
1015        }
1016 
1017    }
1018 
1019 
1020    package br.com.war;
1021 
```

```
1022 public class VotacaoDisputaBean {
1023
1024     public String votar() {
1025         return "confirmacao";
1026     }
1027
1028     public String paginaCadastraCandidato() {
1029         return "candidato";
1030     }
1031
1032     public String paginaCadastraEleicao() {
1033         return "eleicao";
1034     }
1035
1036     public String paginaCadastraVotante() {
1037         return "votante";
1038     }
1039
1040     public String paginaCadastraDisputa() {
1041         return "disputa";
1042     }
1043
1044     public String paginaCadastroInicio() {
1045         return "cadastroInicio";
1046     }
1047
1048 }
1049
1050 package br.com.war;
1051
1052 import javax.faces.application.FacesMessage;
1053 import javax.faces.context.FacesContext;
1054
1055
1056 public class LoginBean {
1057
1058     private String login;
1059     private String senha;
1060
1061     public String getLogin() {
1062         return login;
1063     }
1064
1065     public void setLogin(String login) {
```

```
1066            this.login = login;
1067        }
1068
1069        public String getSenha() {
1070            return senha;
1071        }
1072
1073        public void setSenha(String senha) {
1074            this.senha = senha;
1075        }
1076
1077        public String logar() {
1078        if(login.equals("abimael")){
1079            if(senha.equals("123")){
1080                return "autorizado";
1081            }
1082        }
1083        FacesContext.getCurrentInstance().addMessage("erro", new FacesMessage("Login não
                autorizado"));
1084
1085    return " ";
1086    }
1087
1088        }
1089
1090
1091    package br.com.war;
1092
1093    import javax.faces.application.FacesMessage;
1094    import javax.faces.context.FacesContext;
1095
1096    public class Login {
1097
1098    private boolean loginOk;
1099
1100    private String login;
1101
1102    private String senha;
1103
1104        public boolean isLoginOk() {
1105
1106            return loginOk;
1107
1108        }
```

```
1109
1110    public  String validateLogin(){
1111
1112        if(getLogin()!=null     getSenha()!=null !getLogin().equalsIgnoreCase(getSenha
                ())){
1113
1114            setLoginOk(true);
1115
1116            return "secpage";
1117
1118        }else return "login";
1119
1120    }
1121
1122    public String logar (){
1123        if(login.equalsIgnoreCase("abimael")){
1124            if(senha.equalsIgnoreCase("123")  ){
1125                loginOk = true;
1126                return "logado";
1127            }
1128        }
1129        loginOk = false;
1130         FacesContext.getCurrentInstance().addMessage("erro", new FacesMessage("Login
                 não autorizado"));
1131        return "erro";
1132    }
1133
1134    public void setLogin(String login) {
1135        this.login = login;
1136    }
1137
1138    public String getLogin() {
1139        return login;
1140    }
1141
1142    public void setSenha(String senha) {
1143        this.senha = senha;
1144    }
1145
1146    public String getSenha() {
1147        return senha;
1148    }
1149
1150
```

```
1151
1152        public void setLoginOk(boolean loginOk) {
1153            this.loginOk = loginOk;
1154        }
1155
1156    }
1157
1158
1159    package br.com.war;
1160
1161    import java.util.ArrayList;
1162    import java.util.Date;
1163    import java.util.List;
1164
1165    import javax.faces.model.SelectItem;
1166
1167    import br.com.beam.Candidato;
1168    import br.com.beam.Eleicao;
1169    import br.com.beam.StatusEleicao;
1170    import br.com.beam.TipoAutenticacao;
1171    import br.com.beam.Votante;
1172    import br.com.dao.EleicaoDAO;
1173    import br.com.dao.VotanteDAO;
1174
1175    public class EleicaoBB {
1176        private String nome;
1177        private Date dataInicio;
1178        private Date dataFim;
1179        private TipoAutenticacao tipoAutenticacao;
1180        private StatusEleicao Status;
1181        private EleicaoDAO eleicaoDAO;
1182        private Eleicao eleicao;
1183        private String tipoAutEscolhido;
1184        private String statusEscolhido;
1185        private List<Eleicao> listaEleicoes;
1186        private Long idEleicao;
1187
1188        public EleicaoBB() {
1189            eleicaoDAO = new EleicaoDAO();
1190            eleicao = new Eleicao();
1191        }
1192
1193        public String cadastrarEleicao() {
1194            Eleicao eleicaoBanco = eleicaoDAO.getEleicaoPorNome(this.getNome());
```

```
1195            String retorno = "";
1196            if (eleicaoBanco == null) {
1197                eleicao.setNome(this.getNome());
1198                eleicao.setDataInicio(this.getDataInicio());
1199                eleicao.setDataFim(this.getDataFim());
1200                this.selecionaTipoAutenticacao();
1201                this.selecionaStatus();
1202                eleicaoDAO.cadastrarEleicao(eleicao);
1203                retorno = "eleicaoCadastrada";
1204            } else {
1205                retorno = "erroNomeEleicao";
1206            }
1207            eleicao.setNome(this.getNome());
1208            eleicao.setDataInicio(this.getDataInicio());
1209            eleicao.setDataFim(this.getDataFim());
1210
1211            this.selecionaTipoAutenticacao();
1212            this.selecionaStatus();
1213            eleicaoDAO.cadastrarEleicao(eleicao);
1214            String retorno = "eleicaoCadastrada";
1215
1216            return retorno;
1217        }
1218
1219        public void selecionaStatus() {
1220            if (statusEscolhido.equalsIgnoreCase("0")){
1221                eleicao.setStatus(StatusEleicao.NAO\_INICIADA);}
1222            else if (statusEscolhido.equalsIgnoreCase("1")){
1223                eleicao.setStatus(StatusEleicao.VOTACAO);
1224            }
1225
1226            else if (statusEscolhido.equalsIgnoreCase("2")){
1227                eleicao.setStatus(StatusEleicao.APURACAO\_DIVULGACAO\_DE\_RESULTADOS);
1228            }
1229            else{
1230                eleicao.setStatus(StatusEleicao.FINALIZADA);
1231            }
1232
1233        }
1234
1235        public void selecionaTipoAutenticacao() {
1236            if (tipoAutEscolhido.equalsIgnoreCase("0"))LOGIN\_SENHA);
1237            else if (tipoAutEscolhido.equalsIgnoreCase("1"))
1238                eleicao.setTipoAutenticacao(TipoAutenticacao.CERTIFICADO\_DIGITAL);
```

```
1239            else
1240                eleicao.setTipoAutenticacao(TipoAutenticacao.AMBOS);
1241
1242        }
1243
1244        public List<String> listaTipoAutenticao() {
1245            List<String> listaTipoAutent = new ArrayList<String>();
1246            for (int i = 0; i < 3; i++) {
1247                listaTipoAutent.add("" + i);
1248            }
1249            return listaTipoAutent;
1250        }
1251
1252        public List<SelectItem> getListaTipoAutenticacao() {
1253            List<SelectItem> listaSelectItem = new ArrayList<SelectItem>();
1254            for (int i = 0; i < this.listaTipoAutenticao().size(); i++) {
1255                listaSelectItem.add(new SelectItem(listaTipoAutenticao().get(i)));
1256            }
1257
1258            return listaSelectItem;
1259        }
1260
1261        public List<String> listaStatusString() {
1262            List<String> lista = new ArrayList<String>();
1263            for (int i = 0; i < 4; i++) {
1264                lista.add("" + i);
1265            }
1266            return lista;
1267        }
1268
1269        public List<SelectItem> getListaStatus() {
1270            List<SelectItem> listaSelectItem = new ArrayList<SelectItem>();
1271            for (int i = 0; i < this.listaStatusString().size(); i++) {
1272                listaSelectItem.add(new SelectItem(listaStatusString().get(i)));
1273            }
1274
1275            return listaSelectItem;
1276        }
1277
1278        public List<SelectItem> getListaEleicoesSelectItem() {
1279            List<SelectItem> listaSelectItem = new ArrayList<SelectItem>();
1280            listaEleicoes = eleicaoDAO.getTodasEleicoes();
1281            for (int i = 0; i < listaEleicoes.size(); i++) {
1282                listaSelectItem.add(new SelectItem(listaEleicoes.get(i).getNome()));
```

```
1283            }
1284            return listaSelectItem;
1285        }
1286
1287        public List<Eleicao> getListaEleicoesComStatus() {
1288            EleicaoDAO dao = new EleicaoDAO();
1289            return dao.getTodasEleicoes();
1290        }
1291
1292        public List<SelectItem> getListaEleicoesNaoApuradasSelectItem() {
1293            List<SelectItem> listaSelectItem = new ArrayList<SelectItem>();
1294            StatusEleicao status = StatusEleicao.VOTACAO;
1295            listaEleicoes = eleicaoDAO.getEleicaoPorStatus(status);
1296            for (int i = 0; i < listaEleicoes.size(); i++) {
1297                listaSelectItem.add(new SelectItem(listaEleicoes.get(i).getNome()));
1298            }
1299            return listaSelectItem;
1300        }
1301
1302        public List<Eleicao> getListaEleicoesPorStatus() {
1303            EleicaoDAO dao = new EleicaoDAO();
1304            StatusEleicao status = StatusEleicao.VOTACAO;
1305            return dao.getEleicaoPorStatus(status);
1306        }
1307
1308        public String getNome() {
1309            return nome;
1310        }
1311
1312        public void setNome(String nome) {
1313            this.nome = nome;
1314        }
1315
1316        public Date getDataInicio() {
1317            return dataInicio;
1318        }
1319
1320        public void setDataInicio(Date dataInicio) {
1321            this.dataInicio = dataInicio;
1322        }
1323
1324        public Date getDataFim() {
1325            return dataFim;
1326        }
```

```
1327
1328        public void setDataFim(Date dataFim) {
1329            this.dataFim = dataFim;
1330        }
1331
1332        public TipoAutenticacao getTipoAutenticacao() {
1333            return tipoAutenticacao;
1334        }
1335
1336        public void setTipoAutenticacao(TipoAutenticacao tipoAutenticacao) {
1337            this.tipoAutenticacao = tipoAutenticacao;
1338        }
1339
1340        public StatusEleicao getStatus() {
1341            return Status;
1342        }
1343
1344        public void setStatus(StatusEleicao status) {
1345            Status = status;
1346        }
1347
1348        public void setTipoAutEscolhido(String tipoAutEscolhido) {
1349            this.tipoAutEscolhido = tipoAutEscolhido;
1350        }
1351
1352        public String getTipoAutEscolhido() {
1353            return tipoAutEscolhido;
1354        }
1355
1356        public void setStatusEscolhido(String statusEscolhido) {
1357            this.statusEscolhido = statusEscolhido;
1358        }
1359
1360        public String getStatusEscolhido() {
1361            return statusEscolhido;
1362        }
1363
1364        public void setListaEleicoes(List<Eleicao> listaEleicoes) {
1365            this.listaEleicoes = listaEleicoes;
1366        }
1367
1368        public List<Eleicao> getListaEleicoes() {
1369            return listaEleicoes;
1370        }
```

```
1371
1372        public void setIdEleicao(Long idEleicao) {
1373            this.idEleicao = idEleicao;
1374        }
1375
1376        public Long getIdEleicao() {
1377            return idEleicao;
1378        }
1379
1380    }
1381
1382
1383    package br.com.war;
1384
1385    import java.util.ArrayList;
1386    import java.util.List;
1387
1388    import javax.faces.model.SelectItem;
1389
1390    import br.com.beam.Disputa;
1391    import br.com.beam.Eleicao;
1392    import br.com.beam.Votante;
1393    import br.com.dao.DisputaDAO;
1394    import br.com.dao.EleicaoDAO;
1395    import br.com.dao.VotanteDAO;
1396
1397    public class DisputaBB {
1398
1399
1400        private String nome;
1401        private String descricao;
1402        private DisputaDAO disputaDAO;
1403        private Disputa disputa;
1404        private EleicaoDAO eleicaoDAO;
1405        private String eleicaoSelecionada;
1406
1407
1408        public DisputaBB(){
1409            disputaDAO = new DisputaDAO();
1410            disputa = new Disputa();
1411            eleicaoDAO = new EleicaoDAO();
1412            this.getListaEleicaoSelectItem();
1413        }
1414
```

```
1415    public String cadastrarDisputa(){
1416        disputa.setNome(this.getNome());
1417        disputa.setDescricao(this.descricao);
1418        disputa.setEleicao(eleicaoDAO.getEleicaoPorNome(this.eleicaoSelecionada));
1419        disputaDAO.cadastraDisputa(disputa);
1420        return "disputaCadastrada";
1421    }
1422
1423    /*public Eleicao retornaEleicaoSelecionada(){
1424        eleicaoDAO = new EleicaoDAO();
1425        return eleicaoDAO.getEleicaoPorStatus(status);
1426    }*/
1427
1428
1429    public List<Disputa> getListaDisputas(){
1430        DisputaDAO dao = new DisputaDAO();
1431         List<Disputa> lista = dao.getTodasDisputas();
1432        return lista;
1433    }
1434
1435
1436    public List<SelectItem> getListaEleicaoSelectItem(){
1437        List<SelectItem> listaSelectItem = new ArrayList<SelectItem>();
1438        List<Eleicao> listaEleicoes = eleicaoDAO.getTodasEleicoes();
1439        for (int i=0;i<listaEleicoes.size();i++){
1440            listaSelectItem.add(new SelectItem(listaEleicoes.get(i).getNome()));
1441        }
1442        return listaSelectItem;
1443    }
1444
1445
1446
1447
1448    public String getNome() {
1449        return nome;
1450    }
1451
1452    public void setNome(String nome) {
1453        this.nome = nome;
1454    }
1455
1456    public void setDescricao(String descricao) {
1457        this.descricao = descricao;
1458    }
```

```
1459
1460        public String getDescricao() {
1461            return descricao;
1462        }
1463
1464        public void setEleicaoSelecionada(String eleicaoSelecionada) {
1465            this.eleicaoSelecionada = eleicaoSelecionada;
1466        }
1467
1468        public String getEleicaoSelecionada() {
1469            return eleicaoSelecionada;
1470        }
1471
1472
1473
1474
1475    }
1476
1477
1478    package util.xml;
1479
1480    import java.beans.XMLEncoder;
1481    import java.io.BufferedInputStream;
1482    import java.io.BufferedOutputStream;
1483    import java.io.File;
1484    import java.io.FileInputStream;
1485    import java.io.FileNotFoundException;
1486    import java.io.FileOutputStream;
1487    import java.io.FileReader;
1488    import java.io.IOException;
1489    import java.io.OutputStream;
1490    import java.util.List;
1491    import java.security.Key;
1492
1493    import com.thoughtworks.xstream.XStream;
1494    import com.thoughtworks.xstream.io.xml.Dom4JDriver;
1495
1496    public class GerenciadorXML {
1497
1498        public String criaXML(List objetos){
1499            XStream xstream = new XStream();
1500            return  xstream.toXML(objetos);
1501        }
1502
```

```
1503    public String criaXML(Key objetos){
1504        XStream xstream = new XStream();
1505        return  xstream.toXML(objetos);
1506    }
1507
1508
1509    public void salvaXML(String caminho, String arquivo){
1510        OutputStream outputStream = null;
1511        try {
1512            outputStream = new FileOutputStream(caminho);
1513        } catch (FileNotFoundException e) {
1514            e.printStackTrace();
1515        }
1516        BufferedOutputStream bufferedOutputStream = new BufferedOutputStream(
                outputStream);
1517        try {
1518            bufferedOutputStream.write(arquivo.getBytes());
1519        } catch (IOException e1) {
1520            // TODO Auto-generated catch block
1521            e1.printStackTrace();
1522        }
1523        //XMLEncoder encoder = new XMLEncoder(bufferedOutputStream);
1524        //encoder.writeObject(arquivo);
1525
1526
1527    //  encoder.close();
1528        try {
1529            bufferedOutputStream.close();
1530        } catch (IOException e) {
1531            e.printStackTrace();
1532        }
1533        try {
1534            outputStream.close();
1535        } catch (IOException e) {
1536            e.printStackTrace();
1537        }
1538
1539
1540    }
1541
1542    public List abreXML2 (String arquivo){
1543        XStream xstream = new XStream();
1544        List amigos = (List) xstream.fromXML(arquivo);
1545        return amigos;
```

```
1546        }
1547
1548        public Object abreXML (String arquivo){
1549            XStream xstream = new XStream();
1550
1551            return xstream.fromXML(arquivo);
1552        }
1553
1554         public String abreXML\_caminho(String caminho) {
1555                FileInputStream inputStream = null;
1556                try {
1557                    inputStream = new FileInputStream(caminho);
1558
1559                    BufferedInputStream arquivo = new BufferedInputStream( new
                            FileInputStream(new File(caminho)));
1560
1561                    byte dados[] = new byte[2048];
1562                    arquivo.read(dados, 0, 2048);
1563                    return new String(dados);
1564                } catch (FileNotFoundException ex) {
1565                    return null;
1566                } catch (IOException e) {
1567                    e.printStackTrace();
1568                    return null;
1569                }
1570            }
1571
1572         public boolean deserialize(String path, Object obj){
1573             XStream stream = new XStream(new Dom4JDriver());
1574                FileInputStream fis = null;
1575                try {
1576                    File file = new File(path);
1577                    if (file.isDirectory()) {
1578                        throw new IllegalArgumentException ("The path must point to a
                                file.");
1579                    }
1580                    if (file.exists()) {
1581                        fis = new FileInputStream(path);
1582                        stream.fromXML(fis, obj);
1583                        return true;
1584                    }
1585                    return false;
1586                } catch (Throwable e) {
1587                    return false;
```

```
1588            } finally {
1589                try { if (fis != null) fis.close(); } catch (IOException e) {}
1590            }
1591        }
1592
1593    public Object createClient(String pathFile ) throws Exception {
1594            XStream xstream = new XStream();
1595            FileReader reader = new FileReader(new File(pathFile));
1596           Object obj =xstream.fromXML(reader);
1597            return obj;
1598        }
1599
1600
1601
1602 }
1603
1604
1605 <?xml version='1.0' encoding='utf-8'?>
1606 <!DOCTYPE hibernate-configuration PUBLIC
1607         "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
1608         "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
1609
1610 <hibernate-configuration>
1611
1612     <session-factory>
1613
1614         <!-- Database connection settings -->
1615         <property name="connection.driver\_class">org.postgresql.Driver</property>
1616         <property name="connection.url">jdbc:postgresql:postgres</property>
1617         <property name="connection.username">postgres</property>
1618         <property name="connection.password">postgresql</property>
1619
1620
1621
1622         <!-- JDBC connection pool (use the built-in) -->
1623         <property name="connection.pool\_size">2</property>
1624
1625         <!-- SQL dialect -->
1626         <property name="dialect">org.hibernate.dialect.PostgreSQLDialect</property>
1627
1628         <!-- Enable Hibernate's current session context -->
1629         <property name="current\_session\_context\_class">thread</property>
1630
1631         <!-- Disable the second-level cache  -->
```

```
1632          <property name="cache.provider\_class">org.hibernate.cache.NoCacheProvider</
                  property>
1633
1634          <!-- Echo all executed SQL to stdout -->
1635          <property name="show\_sql">true</property>
1636
1637          <!-- Drop and re-create the database schema on startup
1638          <property name="hbm2ddl.auto">create</property>
1639
1640          <mapping resource="org/hibernate/tutorial/domain/Event.hbm.xml"/>
1641          <mapping resource="org/hibernate/tutorial/domain/Person.hbm.xml"/>
1642
1643      -->
1644          <mapping class="br.com.beam.Eleicao"/>
1645          <mapping class="br.com.beam.Disputa"/>
1646          <mapping class="br.com.beam.Votante"/>
1647          <mapping class="br.com.beam.Candidato"/>
1648          <mapping class="br.com.beam.PinEleicao"/>
1649          <mapping class="br.com.beam.Login\_Senha"/>
1650          <mapping class="br.com.beam.Voto"/>
1651          <mapping class="br.com.beam.VotanteEleicao"/>
1652      </session-factory>
1653
1654  </hibernate-configuration>
1655
1656  \section{Cliente} \label{segapendice:codigoFonte:cliente}
1657
1658  package br.com.interfaces;
1659
1660  import javax.microedition.lcdui.Command;
1661  import javax.microedition.lcdui.CommandListener;
1662  import javax.microedition.lcdui.Displayable;
1663  import javax.microedition.lcdui.Form;
1664  import javax.microedition.lcdui.List;
1665
1666  import br.com.listas.ListaSelecaoDeEleicao;
1667
1668  public class FormularioAutenticandoNoSistema extends Form implements CommandListener{
1669
1670      Command commandProsseguir;
1671
1672      public FormularioAutenticandoNoSistema(String title) {
1673          super(title);
1674          commandProsseguir = new Command("Prosseguir", Command.ITEM, 1);
```

```
1675          this.addCommand(commandProsseguir);
1676          this.setCommandListener(this);
1677      }
1678
1679      public void commandAction(Command comandoEscolhido, Displayable display) {
1680          if (comandoEscolhido == commandProsseguir) {
1681              int numero = 1;
1682              ListaSelecaoDeEleicao listaSelecaoDeEleicao = new ListaSelecaoDeEleicao("
                      Seleção de Eleição", List.EXCLUSIVE, new String[], null);
1683              listaSelecaoDeEleicao.insert(numero - 1, stringPart, null);
1684          }
1685
1686      }
1687
1688  }
1689
1690
1691  package br.com.interfaces;
1692
1693  import javax.microedition.lcdui.Command;
1694  import javax.microedition.lcdui.CommandListener;
1695  import javax.microedition.lcdui.Displayable;
1696  import javax.microedition.lcdui.Form;
1697  import javax.microedition.lcdui.TextField;
1698
1699  import br.com.web.GerenciadorConexao;
1700
1701  public class FormularioLogin extends Form implements CommandListener {
1702
1703      private TextField textFieldLogin, textFieldSenha;
1704      private Command commandConfirmaLogin;
1705      private GerenciadorConexao gerenciadorConexao;
1706
1707      public FormularioLogin(String title, GerenciadorConexao gerenciadorConexao) {
1708          super(title);
1709          System.out
1710                  .println("antes de setar e o valor do gerenciadorDeConexao eh "
1711                          + gerenciadorConexao);
1712          this.setGerenciadorConexao(gerenciadorConexao);
1713          textFieldLogin = new TextField("Login: ", "", 32, TextField.ANY);
1714          this.append(textFieldLogin);
1715          textFieldSenha = new TextField("Senha :", "", 10, TextField.PASSWORD);
1716          this.append(textFieldSenha);
1717          commandConfirmaLogin = new Command("Entrar", Command.ITEM, 1);
```

```
1718            this.addCommand(commandConfirmaLogin);
1719            this.setCommandListener(this);
1720        }
1721
1722        public void commandAction(Command comandoEscolhido, Displayable telaAtual) {
1723            System.out.println("entrou uahsduahsaduhsauhdisauhdi");
1724            System.out
1725                    .println("o valor do comandoEscolhido eh " + comandoEscolhido);
1726            System.out.println("o valor do display eh " + telaAtual);
1727            if (comandoEscolhido == commandConfirmaLogin) {
1728                this.getGerenciadorConexao().loginNoSistema(
1729                        textFieldLogin.getString(), textFieldSenha.getString());
1730
1731            }
1732        }
1733
1734        public void setTextFieldLogin(TextField textFieldLogin) {
1735            this.textFieldLogin = textFieldLogin;
1736        }
1737
1738        public TextField getTextFieldLogin() {
1739            return textFieldLogin;
1740        }
1741
1742        public void setTextFieldSenha(TextField textFieldSenha) {
1743            this.textFieldSenha = textFieldSenha;
1744        }
1745
1746        public TextField getTextFieldSenha() {
1747            return textFieldSenha;
1748        }
1749
1750        public void setCommandConfirmaLogin(Command commandConfirmaLogin) {
1751            this.commandConfirmaLogin = commandConfirmaLogin;
1752        }
1753
1754        public Command getCommandConfirmaLogin() {
1755            return commandConfirmaLogin;
1756        }
1757
1758        public void setGerenciadorConexao(GerenciadorConexao gerenciadorConexao) {
1759            this.gerenciadorConexao = gerenciadorConexao;
1760        }
1761
```

```
1762      public GerenciadorConexao getGerenciadorConexao() {
1763          return gerenciadorConexao;
1764      }
1765
1766
1767
1768  }
1769
1770
1771  package br.com.interfaces;
1772
1773  import javax.microedition.lcdui.Command;
1774  import javax.microedition.lcdui.CommandListener;
1775  import javax.microedition.lcdui.Displayable;
1776  import javax.microedition.lcdui.Form;
1777  import javax.microedition.lcdui.StringItem;
1778
1779  public class FormularioVotacaoConcluida extends Form implements CommandListener {
1780
1781      private StringItem stringItemVotacaoConcluida;
1782
1783      public FormularioVotacaoConcluida(String title) {
1784          //votar();
1785          super(title);
1786          stringItemVotacaoConcluida = new StringItem("Votacao Finalizada",
1787                  "Obrigado!");
1788          this.append(stringItemVotacaoConcluida);
1789      }
1790
1791      public void commandAction(Command arg0, Displayable arg1) {
1792
1793      }
1794
1795  }
1796
1797
1798  package br.com.interfaces.erros;
1799
1800  import javax.microedition.lcdui.Command;
1801  import javax.microedition.lcdui.CommandListener;
1802  import javax.microedition.lcdui.Displayable;
1803  import javax.microedition.lcdui.Form;
1804
1805  public class FormularioDeErroLogin extends Form implements CommandListener{
```

```
1806
1807      private Command commandVoltaDoErroLogin;
1808
1809
1810      public FormularioDeErroLogin(String title){
1811          super(title);
1812          commandVoltaDoErroLogin = new Command("Voltar", Command.BACK, 1);
1813          this.addCommand(commandVoltaDoErroLogin);
1814          this.setCommandListener(this);
1815      }
1816
1817      public void commandAction(Command arg0, Displayable arg1) {
1818          // TODO Auto-generated method stub
1819
1820      }
1821
1822  }
1823
1824
1825  package br.com.listas;
1826
1827  import javax.microedition.lcdui.Command;
1828  import javax.microedition.lcdui.CommandListener;
1829  import javax.microedition.lcdui.Displayable;
1830  import javax.microedition.lcdui.Image;
1831  import javax.microedition.lcdui.List;
1832
1833  public class ListaCandidatosDaDisputa extends List implements CommandListener{
1834
1835      private Command commandEscolheCandidatoEscolhidoDaLista;
1836
1837      public ListaCandidatosDaDisputa(String title, int listType,
1838              String[] stringElements, Image[] imageElements) {
1839          super(title, listType, stringElements, imageElements);
1840          commandEscolheCandidatoEscolhidoDaLista = new Command("Selecionar", Command.
                ITEM, 1);
1841          this.addCommand(commandEscolheCandidatoEscolhidoDaLista);
1842
1843      }
1844
1845      public void commandAction(Command arg0, Displayable arg1) {
1846
1847      }
1848
```

```
1849 }
1850
1851
1852 package br.com.listas;
1853
1854 import javax.microedition.io.Connector;
1855 import javax.microedition.io.HttpConnection;
1856 import javax.microedition.lcdui.Command;
1857 import javax.microedition.lcdui.CommandListener;
1858 import javax.microedition.lcdui.Displayable;
1859 import javax.microedition.lcdui.Image;
1860 import javax.microedition.lcdui.List;
1861
1862 import br.com.vectors.VetorDeDadosEleicao;
1863 import br.com.web.GerenciadorConexao;
1864
1865 public class ListaSelecaoDeEleicao extends List implements CommandListener {
1866
1867     private String nomeEleicaoSelecionada;
1868     private VetorDeDadosEleicao vetorDeDadosEleicao;
1869
1870     public ListaSelecaoDeEleicao(String nome, int listType,
1871             String[] stringElements, Image[] imageElements) {
1872         super(nome, listType, stringElements, imageElements);
1873
1874
1875         System.out.println("antes de deletar ");
1876         this.deleteAll();
1877         System.out.println("depois de deletar ");
1878
1879         this.setTitle(nome);
1880         this.nomeEleicaoSelecionada = nome;
1881
1882         /*for (int i=0;i<vetorDadosEleicao.size();i++){
1883             int cont = 0;
1884             System.out.println("primeira vez dentro do for----");
1885             String[] candidato = (String[]) vetorDadosEleicao.elementAt(i);
1886
1887             System.out.println("vetorDadosCandidatos.elementAt("+i+") eh "+
                     vetorDadosEleicao.elementAt(i));
1888         }*/
1889
1890     }
1891
```

```
1892    public void buscaListaEleicoes(GerenciadorConexao gerenciadorConexao,int votante\
            _id,String valorpin){
1893        System.gc();
1894        gerenciadorConexao.setAcao("lendoListaEleicoes");
1895        gerenciadorConexao.setUrl("http://localhost:"+gerenciadorConexao.
                getNumeroPorta()+"/Servidor\_TCC/conexaoCelularServidor/" +
1896                    "listaEleicoes.jsf?valorpin="+ valorpin + "votante\_id=" + votante\
                        _id);
1897        gerenciadorConexao.iniciaThread();
1898        System.gc();
1899    }
1900
1901
1902    private void lendoListaEleicoes(GerenciadorConexao gerenciadorConexao) {
1903        StringBuffer pesquisa = new StringBuffer();
1904        try {
1905
1906            String url = gerenciadorConexao.getUrl();
1907            HttpConnection conector = (HttpConnection) Connector.open(url);
1908            gerenciadorConexao.setConexaoHttp(conector);
1909
1910            if (gerenciadorConexao.getConexaoHttp().getResponseCode() ==
                    HttpConnection.HTTP\_OK) {
1911                vetorDeDadosEleicao = new VetorDeDadosEleicao();
1912
1913                String[] dados;
1914                System.out.println("executando o preencheListaEleicoes");
1915
1916                vetorDeDadosEleicao.setInputStreamDadosEleicao(gerenciadorConexao.
                        getConexaoHttp().openInputStream());
1917
1918                System.out.println("Http OK");
1919                pesquisa = new StringBuffer();
1920                int numero = 1, caracterLido = vetorDeDadosEleicao.
                        getInputStreamDadosEleicao().read();
1921                while (caracterLido != ') {
1922                    dados = new String[3];
1923                    while (caracterLido != ') {
1924                        // agora encontra o nome dos candidatos
1925                        pesquisa.append((char) caracterLido);
1926                        caracterLido = vetorDeDadosEleicao.getInputStreamDadosEleicao
                                ().read();
1927                    }
1928
```

```
1929                        dados[0] = pesquisa.toString().trim();
1930                        pesquisa.delete(0, pesquisa.length());
1931                        // pesquisa.append((char) caractereLido);
1932                        caracterLido = vetorDeDadosEleicao.getInputStreamDadosEleicao().
                                read();
1933
1934
1935                        vetorDeDadosEleicao.addElement(dados);
1936                        numero++;
1937                    }
1938                    midletCelular.recebeVetorDeDadosDisputas(vetorDeDadosEleicao);
1939
1940                    vetorDeDadosEleicao.getInputStreamDadosEleicao().close();
1941                    System.out.println("Fechada Conexao Preenche Lista Eleições");
1942                }
1943            } catch (Exception e) {
1944                System.out.println("Exceçao encontrada = " + e);
1945            }
1946        }
1947
1948
1949
1950        public void commandAction(Command arg0, Displayable arg1) {
1951            // TODO Auto-generated method stub
1952
1953        }
1954
1955        public void setNomeEleicaoSelecionada(String nomeEleicaoSelecionada) {
1956            this.nomeEleicaoSelecionada = nomeEleicaoSelecionada;
1957        }
1958
1959        public String getNomeEleicaoSelecionada() {
1960            return nomeEleicaoSelecionada;
1961        }
1962
1963
1964
1965    }
1966
1967
1968    package br.com.midlets;
1969
1970    import javax.microedition.lcdui.Command;
1971    import javax.microedition.lcdui.CommandListener;
```

```
1972 import javax.microedition.lcdui.Display;
1973 import javax.microedition.lcdui.Displayable;
1974 import javax.microedition.lcdui.List;
1975 import javax.microedition.midlet.MIDlet;
1976 import javax.microedition.midlet.MIDletStateChangeException;
1977
1978 import br.com.interfaces.FormularioAutenticandoNoSistema;
1979 import br.com.interfaces.FormularioLogin;
1980 import br.com.interfaces.FormularioVotacaoConcluida;
1981 import br.com.interfaces.erros.FormularioDeErroLogin;
1982 import br.com.listas.ListaCandidatosDaDisputa;
1983 import br.com.listas.ListaSelecaoDeEleicao;
1984 import br.com.web.GerenciadorConexao;
1985
1986 public class MidletVotacaoCelular extends MIDlet implements CommandListener {
1987
1988     private Display display;
1989     private FormularioLogin formLogin;
1990     private FormularioDeErroLogin formDeErroLogin;
1991     private FormularioVotacaoConcluida formVotacaoConcluida;
1992     private FormularioAutenticandoNoSistema formAutenticandoNoSistema;
1993     private ListaSelecaoDeEleicao listaSelecaoDeEleicao;
1994     private ListaCandidatosDaDisputa listaCandidatosDaDisputa;
1995     private GerenciadorConexao gerenciadorConexao;
1996
1997
1998     protected void startApp() throws MIDletStateChangeException {
1999         System.out.println("Iniciou a aplicação");
2000         gerenciadorConexao = new GerenciadorConexao(this);
2001         display = Display.getDisplay(this);
2002         this.criarListas();
2003         this.criacaoFormularios();
2004         display.setCurrent(formLogin);
2005
2006     }
2007
2008     private void criacaoFormularios(){
2009         formLogin = new FormularioLogin("Tela de Login", gerenciadorConexao);
2010         setFormAutenticandoNoSistema(new FormularioAutenticandoNoSistema("
                 Autenticando no Sistema"));
2011         setFormDeErroLogin(new FormularioDeErroLogin("Erro no Login"));
2012         setFormVotacaoConcluida(new FormularioVotacaoConcluida("Votacao Concluida"));
2013     }
2014
```

```
2015    private void criarListas(){
2016        setListaSelecaoDeEleicao(new ListaSelecaoDeEleicao("teste", List.EXCLUSIVE,
                new String[], null));
2017        setListaCandidatosDaDisputa(new ListaCandidatosDaDisputa("teste", List.
                EXCLUSIVE, new String[],  null));
2018
2019    }
2020
2021    public void commandAction(Command arg0, Displayable arg1) {
2022        // TODO Auto-generated method stub
2023
2024    }
2025
2026    protected void pauseApp() {
2027        // TODO Auto-generated method stub
2028
2029    }
2030
2031    protected void destroyApp(boolean arg0) throws MIDletStateChangeException {
2032        System.out.println("Encerrou a Aplicação");
2033
2034    }
2035
2036    public void setDisplay(Display display) {
2037        this.display = display;
2038    }
2039
2040    public Display getDisplay() {
2041        return display;
2042    }
2043
2044    public void setFormDeErroLogin(FormularioDeErroLogin formDeErroLogin) {
2045        this.formDeErroLogin = formDeErroLogin;
2046    }
2047
2048    public FormularioDeErroLogin getFormDeErroLogin() {
2049        return formDeErroLogin;
2050    }
2051
2052    public void setFormVotacaoConcluida(FormularioVotacaoConcluida
            formVotacaoConcluida) {
2053        this.formVotacaoConcluida = formVotacaoConcluida;
2054    }
2055
```

```
2056      public FormularioVotacaoConcluida getFormVotacaoConcluida() {
2057          return formVotacaoConcluida;
2058      }
2059
2060      public void setListaSelecaoDeEleicao(ListaSelecaoDeEleicao listaSelecaoDeEleicao)
              {
2061          this.listaSelecaoDeEleicao = listaSelecaoDeEleicao;
2062      }
2063
2064      public ListaSelecaoDeEleicao getListaSelecaoDeEleicao() {
2065          return listaSelecaoDeEleicao;
2066      }
2067
2068      public void setListaCandidatosDaDisputa(ListaCandidatosDaDisputa
              listaCandidatosDaDisputa) {
2069          this.listaCandidatosDaDisputa = listaCandidatosDaDisputa;
2070      }
2071
2072      public ListaCandidatosDaDisputa getListaCandidatosDaDisputa() {
2073          return listaCandidatosDaDisputa;
2074      }
2075
2076      public void setFormAutenticandoNoSistema(FormularioAutenticandoNoSistema
              formAutenticandoNoSistema) {
2077          this.formAutenticandoNoSistema = formAutenticandoNoSistema;
2078      }
2079
2080      public FormularioAutenticandoNoSistema getFormAutenticandoNoSistema() {
2081          return formAutenticandoNoSistema;
2082      }
2083
2084      public void setGerenciadorConexao(GerenciadorConexao gerenciadorConexao) {
2085          this.gerenciadorConexao = gerenciadorConexao;
2086      }
2087
2088      public GerenciadorConexao getGerenciadorConexao() {
2089          return gerenciadorConexao;
2090      }
2091  }
2092
2093
2094  package br.com.vectors;
2095
2096  import java.io.InputStream;
```

```
2097 import java.util.Vector;
2098
2099 public class VetorDeDadosEleicao extends Vector{
2100
2101     private InputStream inputStreamDadosEleicao;
2102
2103     public VetorDeDadosEleicao(){
2104     }
2105
2106     public void setInputStreamDadosEleicao(InputStream inputStreamDadosEleicao) {
2107         this.inputStreamDadosEleicao = inputStreamDadosEleicao;
2108     }
2109
2110     public InputStream getInputStreamDadosEleicao() {
2111         return inputStreamDadosEleicao;
2112     }
2113
2114 }
2115
2116
2117 package br.com.web;
2118
2119 import java.io.InputStream;
2120
2121 import javax.microedition.io.HttpConnection;
2122 import javax.microedition.lcdui.List;
2123
2124 import br.com.listas.ListaSelecaoDeEleicao;
2125 import br.com.midlets.MidletVotacaoCelular;
2126
2127 public class GerenciadorConexao implements Runnable {
2128
2129     private HttpConnection conexaoHttp;
2130     private String url;
2131     private InputStream objInputStream;
2132     private String acao = "";
2133     private String numeroPorta;
2134     private Thread thread;
2135     private StringBuffer pesquisa = new StringBuffer();
2136     private MidletVotacaoCelular midletVotacaoCelular;
2137     private LoginSistema loginSistema;
2138     private ListaSelecaoDeEleicao listaSelecaoDeEleicao;
2139
2140     public GerenciadorConexao(MidletVotacaoCelular midletVotacaoCelular) {
```

```
2141            this.midletVotacaoCelular = midletVotacaoCelular;
2142
2143        }
2144
2145        public void loginNoSistema(String login, String senha) {
2146            this.setNumeroPorta("8443");
2147            loginSistema = new LoginSistema(login, senha, this);
2148
2149
2150        }
2151
2152        public void lerListaSelecaoDeEleicao(){
2153            this.setNumeroPorta("8443");
2154            listaSelecaoDeEleicao = new ListaSelecaoDeEleicao("Seleção De Eleições",List.
                    EXCLUSIVE,new String[],null);
2155            listaSelecaoDeEleicao.buscaListaEleicoes(this, 1, "123");
2156        }
2157
2158        public void run() {
2159            System.gc();
2160            System.out.println("*****************************************");
2161            System.out.println("dentro do metodo RUN");
2162            System.out.println("*****************************************");
2163            System.out.println("acao igual a " + acao);
2164            try {
2165                if (acao.equalsIgnoreCase("loginNoSistema")) {
2166                    getLoginSistema().lendoLoginNoSistema(this);
2167                } else if (acao.equalsIgnoreCase("insereVoto")) {
2168                    // inserirVoto();
2169                } else if (acao.equalsIgnoreCase("lendoListaEleicoes")) {
2170                    lendoListaEleicoes(this);
2171                }
2172                // this.preencheListas();
2173                // this.preencheListaDisputas();
2174
2175                System.gc();
2176            } catch (Exception e) {
2177                System.out.println("Excecao eh " + e);
2178            }
2179        }
2180
2181        public void iniciaThread() {
2182            System.gc();
2183            System.out.println("Iniciando a thread");
```

```
2184            this.thread = new Thread(this);
2185            this.thread.start();
2186            System.out.println("Iniciou a thread");
2187            System.gc();
2188        }
2189
2190        public void setConexaoHttp(HttpConnection conexaoHttp) {
2191            this.conexaoHttp = conexaoHttp;
2192        }
2193
2194        public HttpConnection getConexaoHttp() {
2195            return conexaoHttp;
2196        }
2197
2198        public void setUrl(String url) {
2199            this.url = url;
2200        }
2201
2202        public String getUrl() {
2203            return url;
2204        }
2205
2206        public void setObjInputStream(InputStream objInputStream) {
2207            this.objInputStream = objInputStream;
2208        }
2209
2210        public InputStream getObjInputStream() {
2211            return objInputStream;
2212        }
2213
2214        public void setAcao(String acao) {
2215            this.acao = acao;
2216        }
2217
2218        public String getAcao() {
2219            return acao;
2220        }
2221
2222        public void setNumeroPorta(String numeroPorta) {
2223            this.numeroPorta = numeroPorta;
2224        }
2225
2226        public String getNumeroPorta() {
2227            return numeroPorta;
```

```
2228      }
2229
2230      public void setPesquisa(StringBuffer pesquisa) {
2231          this.pesquisa = pesquisa;
2232      }
2233
2234      public StringBuffer getPesquisa() {
2235          return pesquisa;
2236      }
2237
2238      public void setMidletVotacaoCelular(
2239              MidletVotacaoCelular midletVotacaoCelular) {
2240          this.midletVotacaoCelular = midletVotacaoCelular;
2241      }
2242
2243      public MidletVotacaoCelular getMidletVotacaoCelular() {
2244          return midletVotacaoCelular;
2245      }
2246
2247      public void setLoginSistema(LoginSistema loginSistema) {
2248          this.loginSistema = loginSistema;
2249      }
2250
2251      public LoginSistema getLoginSistema() {
2252          return loginSistema;
2253      }
2254
2255      public void setListaSelecaoDeEleicao(ListaSelecaoDeEleicao listaSelecaoDeEleicao)
              {
2256          this.listaSelecaoDeEleicao = listaSelecaoDeEleicao;
2257      }
2258
2259      public ListaSelecaoDeEleicao getListaSelecaoDeEleicao() {
2260          return listaSelecaoDeEleicao;
2261      }
2262
2263  }
2264
2265
2266
2267  package br.com.web;
2268
2269  import java.io.IOException;
2270  import java.io.InputStream;
```

```
2271
2272 import javax.microedition.io.Connector;
2273 import javax.microedition.io.HttpConnection;
2274
2275 public class InsercaoVoto {
2276
2277     private void inserirVoto(HttpConnection conexaoHttp, InputStream objInputStream,
          String url) {
2278         System.out
2279                 .println
                        ("----------------------------------------------------------------------------------------
                        ;
2280         System.out
2281                 .println("-------------------------entrando no insere voto
                        ----------------------------------------");
2282         System.out
2283                 .println
                        ("----------------------------------------------------------------------------------------
                        ;
2284         System.gc();
2285         try {
2286             conexaoHttp = (HttpConnection) Connector.open(url);
2287             if (conexaoHttp.getResponseCode() == HttpConnection.HTTP\_OK) {
2288                 objInputStream = conexaoHttp.openInputStream();
2289                 objInputStream.close();
2290                 System.gc();
2291             }
2292         } catch (IOException e) {
2293             System.out.println("Erro ao Inserir Voto");
2294         }
2295     }
2296
2297 }
2298
2299
2300 package br.com.web;
2301
2302 import javax.microedition.io.Connector;
2303 import javax.microedition.io.HttpConnection;
2304
2305 public class LoginSistema {
2306
2307     public LoginSistema(String login, String senha,
2308             GerenciadorConexao gerenciadorConexao) {
```

```
2309            System.gc();
2310            gerenciadorConexao.setAcao("loginNoSistema");
2311            gerenciadorConexao
2312                    .setUrl("http://localhost:"
2313                            + gerenciadorConexao.getNumeroPorta()
2314                            + "/Servidor\_TCC/conexaoCelularServidor/listaUsuario.jsf?
                                    login="
2315                            + login + "senha=" + senha);
2316            gerenciadorConexao.iniciaThread();
2317            System.gc();
2318        }
2319
2320        public void lendoLoginNoSistema(GerenciadorConexao gerenciadorConexao) {
2321            try {
2322                String url = gerenciadorConexao.getUrl();
2323                HttpConnection conector = (HttpConnection) Connector.open(url);
2324                gerenciadorConexao.setConexaoHttp(conector);
2325
2326                if (gerenciadorConexao.getConexaoHttp().getResponseCode() ==
                        HttpConnection.HTTP\_OK) {
2327                    System.out.println("fazendo login");
2328                    gerenciadorConexao.setObjInputStream(gerenciadorConexao
2329                            .getConexaoHttp().openInputStream());
2330                    System.out.println("Http OK");
2331                    gerenciadorConexao.setPesquisa(new StringBuffer());
2332                    int caracterLido = gerenciadorConexao.getObjInputStream()
2333                            .read();
2334                    while (caracterLido != ') {
2335                        gerenciadorConexao.getPesquisa()
2336                                .append((char) caracterLido);
2337                        caracterLido = gerenciadorConexao.getObjInputStream()
2338                                .read();
2339                    }
2340                    if (gerenciadorConexao.getPesquisa().length() > 9) {
2341                        gerenciadorConexao.getMidletVotacaoCelular().getDisplay()
2342                                .setCurrent(
2343                                        gerenciadorConexao
2344                                                .getMidletVotacaoCelular()
2345                                                .getFormAutenticandoNoSistema());
2346                        gerenciadorConexao.setAcao("lerListaDisputas");
2347                        gerenciadorConexao.iniciaThread();
2348                    } else {
2349                        gerenciadorConexao.getMidletVotacaoCelular().getDisplay()
2350                                .setCurrent(
```

```
2351                                        gerenciadorConexao
2352                                                .getMidletVotacaoCelular()
2353                                                .getFormDeErroLogin());
2354                    }
2355                    gerenciadorConexao.getObjInputStream().close();
2356                    System.out.println("Fechada conexao login");
2357                    gerenciadorConexao.setAcao("LoginJaEfetuado");
2358                }
2359            } catch (Exception e) {
2360                System.out.println("Exceçao encontrada = " + e);
2361            }
2362            System.gc();
2363        }
2364
2365    }
```

Apêndice B

Código Fonte Rede Misturadores

Código JMixNet [1]

```
1
2  package br.edu.ufsc.labsec.util;
3
4  public class WrongConfigException extends ConfigException {
5
6      public WrongConfigException(String key) {
7          super("Property '" + key + "' was wrongly set in configuration file.");
8      }
9  }
10
11
12 public class MissingConfigException extends ConfigException {
13
14     public MissingConfigException(String key) {
15         super("Property '" + key + "' was not set in configuration file.");
16     }
17 }
18
19 package br.edu.ufsc.labsec.util;
20
21 import java.io.*;
22 import java.util.*;
23
24 public class ConfigManager extends Properties {
25
26     protected String configFileName;
```

[1] Retirado da Dissertação de Mestrado de Fabiano Castro Pereira[FAB 05]

```
27      protected String readValue;
28
29      /**
30       * Creates a config manager loading the specified file.
31       */
32      public ConfigManager(String fileName) throws ConfigException {
33
34          FileInputStream configFile;
35
36          try {
37              configFileName = fileName;
38              configFile = new FileInputStream(fileName);
39              load(configFile);
40              configFile.close();
41          }
42          catch (FileNotFoundException ex) {
43              throw new ConfigFileNotFoundException(configFileName);
44          }
45          catch (IOException ex) {
46              throw new ConfigException("Error reading the configuration file.");
47          }
48
49      }
50
51      /**
52       * Creates a config manager loading the specified config stream.
53       */
54      public ConfigManager(InputStream configStream) throws ConfigException {
55          try {
56              load(configStream);
57          }
58          catch (IOException ex) {
59              throw new ConfigException("Error reading the configuration.");
60          }
61      }
62
63      /**
64       * Stores the configuration back to original file.
65       */
66      public void store() throws IOException {
67          FileOutputStream out = new FileOutputStream(configFileName);
68          store(out, "");
69          out.close();
70      }
```

```
71
72      /**
73       * Stores the configuration with a header back to original file.
74       */
75      public void store(String header) throws IOException {
76          FileOutputStream out = new FileOutputStream(configFileName);
77          store(out, header);
78          out.close();
79      }
80
81      /**
82       * Get a property as an int.
83       */
84      public int getInt(String key) {
85          return Integer.parseInt(getProperty(key));
86      }
87
88      /**
89       * Get a property as an int, returns the default value argument if the property
             is not found
90       */
91      public int getInt(String key, int defaultValue) {
92          readValue = getProperty(key);
93
94          if (readValue == null)
95              return defaultValue;
96          else
97              return Integer.parseInt(readValue);
98      }
99
100     /**
101      * Get a property as a string.
102      */
103     public String getString(String key) {
104         return getProperty(key);
105     }
106
107     /**
108      * Get a property as a string, returns the default value argument if the property
            is not found
109      */
110     public String getString(String key, String defaultValue) {
111         readValue = getProperty(key);
112
```

```
113         if (readValue == null)
114             return defaultValue;
115         else
116             return readValue;
117     }
118 }
119
120 public class ConfigFileNotFoundException extends ConfigException {
121     public ConfigFileNotFoundException(String file) {
122         super("The configuration file \"" + file + "\" was not found.");
123     }
124 }
125
126 public class ConfigException extends Exception {
127
128     public ConfigException(String msg) {
129         super(msg);
130     }
131
132     public ConfigException(String msg, Throwable cause) {
133         super(msg, cause);
134     }
135 }
136
137 public class ReceiverServer {
138
139     public static void main(String[] args) {
140         ServerSocketFactory factory = ServerSocketFactory.getDefault();
141         Socket clientCon;
142         InputStream cliInput;
143         byte[] msgBuf = new byte[4096];
144         ServerSocket server = null;
145         int amountRead;
146
147         System.out.println("Receiver server running.");
148
149         try {
150             server = factory.createServerSocket(2000);
151         }
152         catch (IOException e) {
153             e.printStackTrace();
154         }
155         while (true) {
156             //start server and wait for connection
```

```
157             try {
158                 clientCon = server.accept();
159                 cliInput = clientCon.getInputStream();
160 
161                 //System.out.print("[");
162                 while ((amountRead = cliInput.read(msgBuf)) > 0) {
163                     //System.out.write(msgBuf, 0, amountRead);
164                 }
165                 //System.out.println("]");
166                 clientCon.close();
167             }
168             catch (IOException e1) {
169                 e1.printStackTrace();
170             }
171         }
172     }
173 }
174 
175 public class ServerGUI implements ActionListener {
176 
177     protected static JFrame mainWindow;
178     protected static JLabel lbConnectedClients;
179     protected static JLabel lbRefusedCons;
180     protected static JLabel lbStoredMsg;
181     protected static JLabel lbBadMsgs;
182     protected static JLabel lbPerformance;
183     protected static JButton btService;
184     protected static JButton btExit;
185     protected static JMixNetServer server;
186 
187     protected static Timer timer;
188     protected static final int TIMER\_PERIOD = 1000;
189     protected static int lastMsgCount = 0;
190     protected static long lastPerfMeasure = 0;
191     protected static double elapsedTime;
192     protected static Double performance;
193 
194     public void createComponents(Container contentPane) {
195 
196         Font fonte = new Font("Tahoma", Font.PLAIN, 11);
197         JLabel label;
198 
199         contentPane.setLayout(new GridLayout(0, 2, 5, 5));
200 
```

```
201         label = new JLabel(" Connected Clients:");
202         label.setFont(fonte);
203         contentPane.add(label);
204 
205         lbConnectedClients = new JLabel("0");
206         lbConnectedClients.setFont(fonte);
207         contentPane.add(lbConnectedClients);
208 
209         label = new JLabel(" Refused Connections:");
210         label.setFont(fonte);
211         contentPane.add(label);
212 
213         lbRefusedCons = new JLabel("0");
214         lbRefusedCons.setFont(fonte);
215         contentPane.add(lbRefusedCons);
216 
217         label = new JLabel(" Stored Messages:");
218         label.setFont(fonte);
219         contentPane.add(label);
220 
221         lbStoredMsg = new JLabel("0");
222         lbStoredMsg.setFont(fonte);
223         contentPane.add(lbStoredMsg);
224 
225         label = new JLabel(" Bad Messages:");
226         label.setFont(fonte);
227         contentPane.add(label);
228 
229         lbBadMsgs = new JLabel("0");
230         lbBadMsgs.setFont(fonte);
231         contentPane.add(lbBadMsgs);
232 
233         label = new JLabel(" Performance:");
234         label.setFont(fonte);
235         contentPane.add(label);
236 
237         lbPerformance = new JLabel("0 mgs/s");
238         lbPerformance.setFont(fonte);
239         contentPane.add(lbPerformance);
240 
241         btService = new JButton("Start");
242         btService.addActionListener(this);
243         contentPane.add(btService);
244 
```

```
245         btExit = new JButton("Exit");
246         btExit.addActionListener(this);
247         contentPane.add(btExit);
248 
249         timer = new Timer(TIMER\_PERIOD, this);
250     }
251 
252     public void actionPerformed(ActionEvent e) {
253         Object src = e.getSource();
254 
255         // Service "Start" and "Stop" button
256         if (src == btService) {
257             if (btService.getText().equals("Start")) {
258                 btService.setText("Stop");
259                 server.start();
260                 timer.start();
261             }
262             else {
263                 //accounting
264                 lastPerfMeasure = new Date().getTime();
265                 timer.stop();
266 
267                 //finish
268                 btService.setText("Start");
269                 server.finishService();
270             }
271         }
272         // "Exit" button
273         else if (src == btExit) {
274             System.exit(0);
275         }
276         // Timer
277         else if (src == timer) {
278             //measure performance
279             elapsedTime = (new Date().getTime() - lastPerfMeasure) / 1000.0;
280             lastPerfMeasure = new Date().getTime();
281             performance = new Double((server.getProcessedMsgCount() - lastMsgCount) /
                    elapsedTime);
282             lastMsgCount = server.getProcessedMsgCount();
283             lbPerformance.setText(performance.intValue() + " msg/s");
284 
285             //other stats
286             //UNDO lbBadMsgs.setText(server.getBadMsgCount() + "");
287             lbStoredMsg.setText(server.getStoredMsgCount() + "");
```

```
288             lbConnectedClients.setText(server.getConnectedClientes() + "");
289             lbRefusedCons.setText(server.getRefusedConnections() + "");
290         }
291     }
292
293     private static void createAndShowGUI() {
294         //create and set up the window
295         mainWindow = new JFrame("JMixNet Server");
296         mainWindow.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);
297         mainWindow.setLocation(500,300);
298         mainWindow.setResizable(false);
299
300         ServerGUI serverGUI = new ServerGUI();
301         serverGUI.createComponents(mainWindow.getContentPane());
302
303         //Display the window.
304         mainWindow.pack();
305         mainWindow.setVisible(true);
306     }
307
308     public static void main(String[] args) throws FileNotFoundException,
            JMixNetException {
309         JMixNetServer server;
310
311         //create JMixNet server according to command line args
312         if (0 == args.length)
313             server = new JMixNetServer();
314         else
315             server = new JMixNetServer(new FileInputStream(args[0]));
316
317         //creating and showing GUI.
318         javax.swing.SwingUtilities.invokeLater(new Runnable() {
319             public void run() {
320                 createAndShowGUI();
321             }
322         });
323         ServerGUI.setServer(server);
324     }
325
326     public static void setServer(JMixNetServer server) {
327         ServerGUI.server = server;
328     }
329 }
330
```

```
331 public interface MessageDeliverer {
332     public void deliverMsg(ByteBuffer msgBuffer);
333 }
334
335 public class MessageBatch {
336
337     //buffers-related
338     protected LinkedList emptyBuffers;
339     protected LinkedList fullBuffers;
340     protected LinkedList flushBuffers;
341     protected ByteBuffer anEmptyBuffer;
342     protected ByteBuffer aFullBuffer;
343     protected int allocationCount = 0;
344
345     //configuration-related
346     protected int msgSize;
347     protected int poolSize;
348     protected int flushPercent;
349
350     //flush-related
351     protected int numChosenMsg;
352     protected Iterator it;
353     protected ByteBuffer currentBuffer;
354     protected int i;
355
356     /**
357      * Creates the batch.
358      *
359      * @param batchSize Size of the batch.
360      * @param poolSize Size of the pool of messages to keep on each round.
361      * @param flushPercent Percentage of message to be sent on each round.
362      * @param msgSize Size of the messages contained by the batch.
363      */
364     public MessageBatch(int initialBatchSize, int poolSize, int flushPercent, int
         msgSize) {
365         emptyBuffers = new LinkedList();
366         fullBuffers = new LinkedList();
367         ByteBuffer newMsgBuffer;
368
369         this.msgSize = msgSize;
370         this.poolSize = poolSize;
371         this.flushPercent = flushPercent;
372
373         //create message buffers
```

```
374             for (int i = 0; i < initialBatchSize; i++) {
375                 newMsgBuffer = ByteBuffer.allocate(msgSize);
376     if (++allocationCount % 20 == 0) System.out.println("CONTRUCTOR up to now: " +
            allocationCount); //UNDO
377                 emptyBuffers.add(newMsgBuffer);
378             }
379         }
380
381         /**
382          * Returns an empty buffer.
383          */
384         public ByteBuffer getEmptyBuffer() {
385             if (emptyBuffers.size() > 0) {
386                 //take from emptyBuffers
387                 it = emptyBuffers.iterator();
388                 anEmptyBuffer = (ByteBuffer)it.next();
389                 it.remove();
390             }
391             else {
392                 //allocate new
393                 anEmptyBuffer = ByteBuffer.allocate(msgSize);
394     if (++allocationCount % 20 == 0) System.out.println("GETEMPTY up to now: " +
            allocationCount); //UNDO
395             }
396             return anEmptyBuffer;
397         }
398
399         /**
400          * Add the new buffer to the full buffers set.
401          *
402          * @param newBuffer The buffer to be added.
403          */
404         public void addFullBuffer(ByteBuffer newBuffer) {
405             fullBuffers.add(newBuffer);
406         }
407
408         /**
409          * Add the new buffer to the empty buffers set.
410          *
411          * @param newBuffer The buffer to be added.
412          */
413         public void addEmptyBuffer(ByteBuffer newBuffer) {
414             emptyBuffers.add(newBuffer);
415         }
```

```
416
417     /**
418      * Add all buffers passed in to the empty buffers set.
419      *
420      * @param buffers The buffers to be added.
421      */
422     public void addEmptyBuffers(Collection buffers) {
423         emptyBuffers.addAll(buffers);
424     }
425
426     /**
427      * Verify if there are enough messages to make a flush.
428      */
429     public boolean isReadyToFlush() {
430         //if there is at least one message over the pool size, it is ready to flush
431         return fullBuffers.size() - poolSize >= 1;
432     }
433
434     /**
435      * Returns the selected messages to be flushed.
436      */
437     public LinkedList getFlushList() {
438         flushBuffers = new LinkedList();
439
440         //number of chosen messages to be flushed
441         numChosenMsg = (new Double(flushPercent * (fullBuffers.size() - poolSize) /
                100.0)).intValue();
442
443         //at least one message must be flushed
444         if (numChosenMsg == 0) numChosenMsg = 1;
445
446         //mix and leave just the chosen messages
447         Collections.shuffle(fullBuffers);
448         Object obj;
449         for (i = 0; i < numChosenMsg; i++) {
450             obj = fullBuffers.removeFirst();
451             flushBuffers.add(obj);
452         }
453         return flushBuffers;
454     }
455
456     /**
457      * Get all full buffers.
458      *
```

```
459         * @return all full buffers.
460         */
461        public LinkedList getWholeList() {
462            return new LinkedList(fullBuffers);
463        }
464    }
465
466    public class JMixNetServer extends Thread {
467
468        //configuration-related
469        protected JMixNetConfigManager config;
470
471        //message handling
472        protected int messageSize;
473        protected MessageBatch messageBatch;
474        protected int sessionKeyFieldSize;
475        protected final int sessionKeySize = 16; //AES
476        protected final int parametersSize = 18; //AES parameters
477        protected final int hashFieldSize = 16; //MD5
478        protected byte[] receivedHash = new byte[hashFieldSize];
479        protected String strReceivedHash;
480        protected byte[] plainData;
481        protected Iterator iterator;
482        protected int flushPeriod;
483        protected long lastFlush;
484        protected boolean isGoodMessage;
485        protected HashMap receivedMsgTrack = new HashMap();
486        protected int maxMsgTrackSize;
487
488        //accounting
489        protected int connectedClients = 0;
490        protected int refusedConnections;
491        protected int badMsgCount;
492        protected int processedMsgCount;
493
494        //flush-related
495        protected ByteBuffer flushMessage;
496        protected LinkedList flushList;
497        protected DelivererThread delivererThread;
498
499        //reports
500        protected PrintStream errorReport = System.err;
501        protected PrintStream serverReport = System.out;
502
```

```
503     //cryptographers and hash
504     protected SymCryptographer symCrypto;
505     protected AsymCryptographer asymCrypto;
506     protected MessageDigest digester;
507
508     //to wait for previous server
509     protected SSLServerSocketFactory serverSocketFactory = null;
510     protected SSLServerSocket serverSocket = null;
511     protected SSLSocket previousServerSocket = null;
512
513     //to connect to the next server
514     protected SSLSocketFactory socketFactory = null;
515     protected SSLSocket nextServerSocket = null;
516
517     //to receive and send messages
518     protected InputStream previousServerInput = null;
519     protected OutputStream nextServerOutput = null;
520     protected boolean serviceActive = true;
521     protected Selector selector = null;
522
523     //to receive client connections
524     protected ServerSocketChannel serverChannel = null;
525     protected SocketChannel clientSocket = null;
526     protected HashMap clientBuffers = new HashMap();
527     protected HashMap allowedClients;
528
529     /**
530      * Construct the server with initial setup.
531      */
532     public JMixNetServer() throws JMixNetException {
533         try {
534             config = new JMixNetConfigManager();
535         }
536         catch (ConfigException ex) {
537             showError("Config error occurred:" + ex.getMessage());
538             throw new JMixNetException(ex);
539         }
540         configureServer();
541     }
542
543     /**
544      * Construct the server with initial setup, based on specified config file.
545      *
546      * @param configFile Configuration file other than "jmixnet.cfg"
```

```
547         */
548        public JMixNetServer(InputStream configStream) throws JMixNetException {
549            try {
550                config = new JMixNetConfigManager(configStream);
551            }
552            catch (ConfigException ex) {
553                showError("Config error occurred:" + ex.getMessage());
554                throw new JMixNetException(ex);
555            }
556            configureServer();
557        }
558
559        /**
560         * Configurations based on config file.
561         */
562        protected void configureServer() throws JMixNetException {
563            //ensure Bouncy Castle Provider is available
564            if (Security.getProvider("BC") == null) {
565                Security.addProvider(new BouncyCastleProvider());
566            }
567            //SSL-related settings
568            System.setProperty("javax.net.ssl.keyStore", config.getKeyStoreLocation());
569            System.setProperty("javax.net.ssl.keyStorePassword", config.
                    getKeyStorePassword());
570
571            //message batch config
572            messageSize = config.getMessageSize();
573            messageBatch = new MessageBatch(config.getMessageBatchSize(),
574                config.getPoolSize(), config.getFlushPercent(), messageSize);
575            flushPeriod = config.getFlushPeriod();
576            plainData = new byte[messageSize];
577            maxMsgTrackSize = config.getMaxMsgTrackSize();
578
579            //define allowed clients
580            allowedClients = config.getAllowedClients();
581
582            //cryptography services and hash
583            try {
584                symCrypto = new SymCryptographer();
585                asymCrypto = new AsymCryptographer(config.getKeyStoreLocation(), config.
                        getKeyStorePassword());
586                asymCrypto.useAlias("serverKey", null);
587                sessionKeyFieldSize = X509CertificateInfo.getKeyBytesSize((
                        X509Certificate)asymCrypto.getCertificate());
```

```
588             digester = MessageDigest.getInstance("MD5");
589         }
590         catch (CryptoException ex) {
591             showError("Unable to use cryptography services:" + ex.getMessage());
592             throw new JMixNetException(ex);
593         }
594         catch (NoSuchAlgorithmException ex) {
595             showError("Unable to get a message digest instance:" + ex.getMessage());
596             throw new JMixNetException(ex);
597         }
598     }
599 
600     /**
601      * Uses error report to display desired error message.
602      *
603      * @param message Error message to be shown.
604      */
605     protected void showError(String message) {
606         errorReport.println("ERROR: " + message);
607     }
608 
609     /**
610      * Uses server report to display desired log message.
611      *
612      * @param message Log message to be shown.
613      */
614     protected void showLog(String message) {
615         serverReport.println("Log: " + message);
616     }
617 
618     /**
619      * Set the PrintStream object to be used as the error report.
620      *
621      * @param report PrintStream object to be used as the error report.
622      */
623     public void setErrorReport(PrintStream report) {
624         errorReport = report;
625     }
626 
627     /**
628      * Set the PrintStream object to be used as the server report.
629      *
630      * @param report PrintStream object to be used as the server report.
631      */
```

```
632     public void setServerReport(PrintStream report) {
633         serverReport = report;
634     }
635 
636     /**
637      * Based on the mixnet chain configuration, connects to the next mix.
638      */
639     protected void connectToNextServer() throws JMixNetException {
640         showLog("Connecting to next server on port " + String.valueOf(config.
                getNextServerPort()) + "...");
641         try {
642             //stablish connection
643             nextServerSocket = (SSLSocket)socketFactory.createSocket(config.
                    getNextServer(), config.getNextServerPort());
644             nextServerOutput = nextServerSocket.getOutputStream();
645             showLog("Connected to next server.");
646 
647             //starts deliverer thread
648             delivererThread = new DelivererThread();
649             delivererThread.setServerOutput(nextServerOutput);
650             delivererThread.start();
651         }
652         catch (UnknownHostException ex) {
653             showError("Unable to connect to the next server. The host \"" + config.
                    getNextServer() + "\" is unknown.");
654             throw new JMixNetException(ex);
655         }
656         catch (ConnectException ex) {
657             showError("The next server is not available:" + ex.getMessage());
658             throw new JMixNetException(ex);
659         }
660         catch (IOException ex) {
661             showError("An IO error occurred when trying to connect to the next server
                    :" + ex.getMessage());
662             throw new JMixNetException(ex);
663         }
664     }
665 
666     /**
667      * Based on the mixnet chain configuration, waits for the connection of the
             previous mix.
668      */
669     protected void waitPreviousServer() throws JMixNetException {
670         try {
```

```
671             boolean connected = false;
672             while (!connected) {
673                 showLog("Waiting on port " + config.getPort() + " for previous server
                        connection...");
674                 previousServerSocket = (SSLSocket)serverSocket.accept();
675
676                 //verify if the connected peer is the desired one
677 // UNDO                 if (previousServerSocket.getSession().getPeerHost().equals(
        config.getPreviousServer())) {
678                     connected = true;
679                     showLog("Previous server connected.");
680                     previousServerInput = previousServerSocket.getInputStream();
681 /*                 }
682                 else {
683                     showLog("Refused connection from '" + previousServerSocket.
                            getSession().getPeerHost() + "'.");
684                     previousServerSocket.close();
685                 }*/
686             }
687         }
688         catch (IOException ex) {
689             showError("Error waiting for previous server connection:" + ex.getMessage
                    ());
690             throw new JMixNetException(ex);
691         }
692     }
693
694     /**
695      * Get the messages ready to be delivered and pass them to the delivererThread
             thread.
696      *
697      * @param msgList The messages to be delivered.
698      */
699     protected void deliverMessages(LinkedList msgList) {
700         //set new messages to be delivered and get the handled buffers
701         msgList = delivererThread.exchangeBuffers(msgList);
702         messageBatch.addEmptyBuffers(msgList);
703
704         //free
705         msgList.clear();
706         msgList = null;
707     }
708
709     /**
```

```
710         * Verifies if a new client message is repeated, discarding it (if true),
711         * or adding this to the message batch (if false).
712         * This is used only by the first server.
713         *
714         * @param newMsg Message just received.
715         */
716        protected void handleClientMessage(ByteBuffer newMsg) {
717            isGoodMessage = true;
718            try {
719                //take the session key and cipher parameters (uses plainData as temporary
                        buffer)
720                asymCrypto.decrypt(newMsg.array(), 0, sessionKeyFieldSize, plainData);
721                symCrypto.setEncodedKey(plainData, 0, sessionKeySize);
722                symCrypto.setEncodedParameters(plainData, sessionKeySize, parametersSize)
                        ;
723
724                //decrypt data
725                symCrypto.decrypt(newMsg.array(), sessionKeyFieldSize, messageSize -
                        sessionKeyFieldSize, plainData);
726
727                //compute and compare hashes
728                digester.update(plainData, hashFieldSize, messageSize -
                        sessionKeyFieldSize - hashFieldSize);
729                System.arraycopy(plainData, 0, receivedHash, 0, hashFieldSize);
730                isGoodMessage = MessageDigest.isEqual(digester.digest(), receivedHash);
731
732                //verify message repeat
733                strReceivedHash = new String(receivedHash);
734                if (receivedMsgTrack.containsKey(strReceivedHash)) {
735                    showLog("Repeated message received.");
736                    badMsgCount++;
737                    isGoodMessage = false;
738                }
739                else {
740                    if (receivedMsgTrack.size() > maxMsgTrackSize) {
741                        receivedMsgTrack.clear();
742                    }
743                    receivedMsgTrack.put(strReceivedHash, null);
744                }
745            }
746            catch (Exception ex) {
747                showError("Error handling client message." + ex.getMessage());
748                isGoodMessage = false;
749            }
```

```
750             newMsg.clear();
751             if (isGoodMessage) {
752                 //leave the message ready to be sent to the next server
753                 newMsg.put(plainData, hashFieldSize, plainData.length - hashFieldSize).
                        clear();
754                 messageBatch.addFullBuffer(newMsg);
755                 processedMsgCount++;
756 
757                 //verify flush condition
758                 if (messageBatch.isReadyToFlush()  ((new Date()).getTime() - lastFlush >
                        flushPeriod)) {
759                     deliverMessages(messageBatch.getFlushList());
760                     lastFlush = (new Date()).getTime();
761                 }
762             }
763             else {
764                 //treat the bad message as an empty buffer
765                 messageBatch.addEmptyBuffer(newMsg);
766                 badMsgCount++;
767                 showLog("Bad message received.");
768             }
769         }
770 
771         /**
772          * Decrypts the message and add it to the message batch.
773          * This is used only by middle or last servers.
774          *
775          * @param newMsg Message just received.
776          */
777         protected void handleServerMessage(ByteBuffer newMsg) {
778             try {
779                 //take the session key and cipher parameters (uses plainData as temporary
                        buffer)
780                 asymCrypto.decrypt(newMsg.array(), 0, sessionKeyFieldSize, plainData);
781                 symCrypto.setEncodedKey(plainData, 0, sessionKeySize);
782                 symCrypto.setEncodedParameters(plainData, sessionKeySize, parametersSize)
                        ;
783 
784                 //decrypt data and put the plain data into the plaindata array
785                 symCrypto.decrypt(newMsg.array(), sessionKeyFieldSize, messageSize -
                        sessionKeyFieldSize, plainData);
786 
787                 //leave the message ready to be sent
788                 newMsg.clear();
```

```
789             newMsg.put(plainData).clear();
790             messageBatch.addFullBuffer(newMsg);
791
792             //verify flush condition
793             if (messageBatch.isReadyToFlush()  ((new Date()).getTime() - lastFlush >
                    flushPeriod)) {
794                 //flush
795                 deliverMessages(messageBatch.getFlushList());
796                 lastFlush = (new Date()).getTime();
797             }
798             processedMsgCount++;
799         }
800         catch (Exception ex) {
801             messageBatch.addEmptyBuffer(newMsg);
802             badMsgCount++;
803             //UNDO showError("Error handling server message." + ex.getMessage());
804         }
805     }
806
807     /* (non-Javadoc)
808      * @see java.lang.Runnablerun()
809      */
810     public void run() {
811         ByteBuffer msgBuffer;
812
813         showLog("Server started.");
814         try {
815             //connect to next server if this is not the last
816             if (config.getNextServer() != null) {
817                 socketFactory = (SSLSocketFactory)SSLSocketFactory.getDefault();
818                 connectToNextServer();
819             }
820             //verify if this is the last, or a middle server
821             if (config.getPreviousServer() != null) {
822
823                 //open port and wait for previous server
824                 serverSocketFactory = (SSLServerSocketFactory)SSLServerSocketFactory.
                        getDefault();
825                 try {
826                     serverSocket = (SSLServerSocket)serverSocketFactory.
                            createServerSocket(config.getPort());
827                 }
828                 catch (IOException ex) {
829                     showError("Unable to create server socket:" + ex.getMessage());
```

```
830                     throw new JMixNetException(ex);
831                 }
832                 waitPreviousServer();
833 
834                 //verify server position
835                 if (config.getNextServer() == null) {
836                     //create delivererThread thread
837                     delivererThread = new DelivererThread(config.getMsgDelivererClass
                            ());
838                     delivererThread.start();
839                     showLog("MixNet service active as the LAST node.");
840                 }
841                 else {
842                     showLog("MixNet service active as a MIDDLE node.");
843                 }
844 
845                 boolean messageReceived;
846                 int amountRead;
847                 int totalRead;
848 
849                 //wait for messages from previous server while service is active
850                 while (serviceActive) {
851                     messageReceived = false;
852                     totalRead = 0;
853                     msgBuffer = messageBatch.getEmptyBuffer();
854 
855                     //wait whole message
856                     while (!messageReceived) {
857                         try {
858                             amountRead = previousServerInput.read(msgBuffer.array(),
                                    totalRead, messageSize - totalRead);
859                         }
860                         catch (IOException ex) {
861                             //verify if this was a connection reset or an error
862                             if (ex.getMessage().equals("Connection reset")) {
863                                 showLog("Previous server disconnected.");
864                             }
865                             else {
866                                 showError("Error reading from previous server socket:
                                        " + ex.getMessage());
867                             }
868 
869                             //clear buffer
870                             msgBuffer.clear();
```

```
871                         messageBatch.addEmptyBuffer(msgBuffer);
872
873                         waitPreviousServer();
874                         break;
875                     }
876                     totalRead += amountRead;
877                     messageReceived = totalRead == messageSize;
878                 }
879                 //handle message
880                 if (messageReceived) {
881                     handleServerMessage(msgBuffer);
882                 }
883             }
884             try {
885                 serverSocket.close();
886             }
887             catch (IOException ex) {
888                 showError("Unable to close server socket:" + ex.getMessage());
889             }
890         }
891         //wait for client connections, as we are the first server
892         else {
893             SocketChannel channel = null;
894             SelectionKey key = null;
895             Iterator it = null;
896
897             //open a non-blocking service
898             try {
899                 serverChannel = ServerSocketChannel.open();
900                 serverChannel.socket().bind(new InetSocketAddress(config.getPort
                        ()));
901                 serverChannel.configureBlocking (false);
902                 selector = Selector.open();
903             }
904             catch (IOException ex) {
905                 if (null == serverChannel)
906                     showError("Unable to open server channel:" + ex.getMessage())
                            ;
907                 else if (!serverChannel.socket().isBound())
908                     showError("Unable to listen for clients. Por already in use:
                            " + String.valueOf(config.getPort()));
909                 throw new JMixNetException(ex);
910             }
911             int amountRead;
```

```
912
913                 //accept new connections thru selector
914                 serverChannel.register(selector, SelectionKey.OP\_ACCEPT);
915
916                 //keep waiting while the service is active
917                 showLog("MixNet service active as the FIRST node.");
918                 while (serviceActive) {
919                     //if we have nothing to do, back and test if service is still
                            active
920                     try {
921                         if (selector.select() == 0) {
922                             continue;
923                         }
924                     }
925                     catch (IOException ex) {
926                         showError("Unable to select channels:" + ex.getMessage());
927                     }
928                     it = selector.selectedKeys().iterator();
929
930                     //handle each key
931                     while (it.hasNext()) {
932                         key = (SelectionKey) it.next();
933
934                         //verify if this is a new connection
935                         if (key.isAcceptable()) {
936                             try {
937                                 channel = serverChannel.accept();
938                             }
939                             catch (IOException ex) {
940                                 showError("Unable to accept client connection:" + ex.
                                        getMessage());
941                             }
942                             //verify if connection is still active
943                             if (channel != null) {
944                                 //allow only config file listed IPs
945                                 if ((allowedClients.size() > 0)  !allowedClients.
                                        containsKey(channel.socket().getInetAddress().
                                        getHostAddress())) {
946                                     try {
947                                         showLog("Connection from '" + channel.socket
                                                ().getInetAddress().getCanonicalHostName
                                                () + "' refused.");
948                                         refusedConnections++;
949                                         channel.close();
```

```
950                         }
951                         catch (IOException ex) {
952                             showError("Unable to close client channel:" +
                                    ex.getMessage());
953                         }
954                     }
955                     //allow only one active connection per IP address
956                     else if (!clientBuffers.containsKey(channel.socket().
                        getInetAddress())) {
957                         try {
958                             //define the buffer this client will write to
959                             clientBuffers.put(channel.socket().
                                    getInetAddress(), messageBatch.
                                    getEmptyBuffer());
960
961                             //setup channel
962                             channel.configureBlocking (false);
963                             channel.register(selector, SelectionKey.OP\
                                    _READ);
964                             showLog("Client '" + channel.socket().
                                    getInetAddress().getCanonicalHostName() +
                                     "' connected.");
965                             connectedClients++;
966                         }
967                         catch (IOException ex) {
968                             showError("Unable to set new connection
                                    channel to non-blocking:" + ex.getMessage
                                    ());
969                         }
970                     }
971                     else {
972                         try {
973                             showLog("Extra client connection from '" +
                                    channel.socket().getInetAddress().
                                    getCanonicalHostName() + "' refused.");
974                             refusedConnections++;
975                             channel.close();
976                         }
977                         catch (IOException ex) {
978                             showError("Unable to close client channel:" +
                                    ex.getMessage());
979                         }
980                     }
981                 }
```

```
982                     }
983                     //verify if this is data to be read
984                     if (key.isReadable()) {
985                         channel = (SocketChannel) key.channel();
986                         msgBuffer = (ByteBuffer) clientBuffers.get(channel.socket
                                ().getInetAddress());
987
988                         //after a whole message is received, the buffer must be
                                taken again
989                         if (msgBuffer == null) {
990                             msgBuffer = messageBatch.getEmptyBuffer();
991                             clientBuffers.put(channel.socket().getInetAddress(),
                                    msgBuffer);
992                         }
993                         try {
994                             if ((amountRead = channel.read(msgBuffer)) > 0) {
995                                 //verify if the whole message was read
996                                 if (msgBuffer.position() == messageSize) {
997                                     //handle message
998                                     msgBuffer.flip();
999                                     handleClientMessage(msgBuffer);
1000                                    clientBuffers.remove(channel.socket().
                                            getInetAddress());
1001                                }
1002                            }
1003                            if (amountRead < 0) {
1004                                //on EOF, remove client and close channel
1005                                showLog("Client '" + channel.socket().
                                        getInetAddress().getCanonicalHostName() + "'
                                        disconnected.");
1006                                connectedClients--;
1007                                clientBuffers.remove(channel.socket().
                                        getInetAddress());
1008                                channel.close();
1009                            }
1010                        }
1011                        catch (IOException ex) {
1012                            showError("Unable to communicate with client:" + ex.
                                    getMessage());
1013                            showLog("Client '" + channel.socket().getInetAddress
                                    ().getCanonicalHostName() + "' disconnected.");
1014                            connectedClients--;
1015                            clientBuffers.remove(channel.socket().getInetAddress
                                    ());
```

```
1016                                try {channel.close();}
1017                                catch (IOException e) {
1018                                    showError("Unable to close client channel:" + ex.
                                            getMessage());
1019                                }
1020                            }
1021
1022                        }
1023                        it.remove( );
1024                    }
1025                }
1026            }
1027            showLog("MixNet service finished.");
1028
1029        }
1030        catch (ClosedByInterruptException ex) {
1031            showLog("MixNet service finished through interruption.");
1032        }
1033        catch (ClosedChannelException ex) {
1034            ex.printStackTrace();
1035        }
1036        catch (JMixNetException ex) {
1037            showError(ex.getMessage());
1038        }
1039    }
1040
1041    /**
1042     * Finishes the service.
1043     */
1044    public void finishService() {
1045        //flush remaining messages
1046        deliverMessages(messageBatch.getWholeList());
1047        serviceActive = false;
1048
1049        //verify if we are the first server
1050        if (config.getPreviousServer() == null)
1051            selector.wakeup();
1052
1053        this.interrupt();
1054    }
1055
1056    public static void main(String[] args) throws JMixNetException,
            FileNotFoundException {
1057        JMixNetServer server;
```

```
1058            BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
1059
1060            //start JMixNet server according to command line args
1061            if (0 == args.length)
1062                server = new JMixNetServer();
1063            else
1064                server = new JMixNetServer(new FileInputStream(args[0]));
1065
1066            server.start();
1067
1068            try {
1069                //wait for console command
1070                String command = reader.readLine();
1071                while (!command.equals("exit")) {
1072                    //handle command
1073                    System.out.println("Unknown command: [" + command + "]");
1074
1075                    //read the next command
1076                    command = reader.readLine();
1077                }
1078            }
1079            catch (IOException ex) {
1080                ex.printStackTrace();
1081            }
1082            server.finishService();
1083
1084            System.out.println("Main thread finished.");
1085        }
1086
1087        /**
1088         * @return Returns the number of bad messages received.
1089         */
1090        public int getBadMsgCount() {
1091            return badMsgCount;
1092        }
1093
1094        /**
1095         * @return Returns the number of connected clientes.
1096         */
1097        public int getConnectedClientes() {
1098            return connectedClients;
1099        }
1100
1101        /**
```

```
1102         * @return Returns the number of processed messages.
1103         */
1104        public int getProcessedMsgCount() {
1105            return processedMsgCount;
1106        }
1107
1108        /**
1109         * @return Returns the number of refused connections.
1110         */
1111        public int getRefusedConnections() {
1112            return refusedConnections;
1113        }
1114
1115        /**
1116         * @return Returns the current number of messages stored in the batch.
1117         */
1118        public int getStoredMsgCount() {
1119            return messageBatch.fullBuffers.size();
1120        }
1121    }
1122
1123    public class JMixNetException extends Exception {
1124
1125        public JMixNetException() {
1126            super("Unsolvable problem encountered.");
1127        }
1128
1129        public JMixNetException(String message) {
1130            super(message);
1131        }
1132
1133        public JMixNetException(Throwable cause) {
1134            super(cause);
1135        }
1136
1137        public JMixNetException(String msg, Throwable cause) {
1138            super(msg, cause);
1139        }
1140    }
1141
1142
1143    public class JMixNetDeliverer implements MessageDeliverer {
1144
1145        protected PrintStream errorReport = System.err;
```

```
1146      protected InetAddress receiver;
1147      protected byte[] addressBuf = new byte[4];
1148      protected short receiverPort;
1149      protected short meaningfulMsgSize;
1150      protected Socket receiverSocket;
1151
1152      public void deliverMsg(ByteBuffer msgBuffer) {
1153          try {
1154              //get the meaningful message size
1155              meaningfulMsgSize = msgBuffer.getShort();
1156
1157              //get the raw address and port
1158              msgBuffer.get(addressBuf, 0, 4);
1159              receiver = InetAddress.getByAddress(addressBuf);
1160              receiverPort = msgBuffer.getShort();
1161
1162              //connect and deliver
1163              receiverSocket = new Socket(receiver, receiverPort);
1164              receiverSocket.getOutputStream().write(msgBuffer.array(), 8,
                      meaningfulMsgSize - 6);
1165              receiverSocket.close();
1166          }
1167          catch (UnknownHostException ex) {
1168              errorReport.println("DELIVERY ERROR - Unknown host: " + receiver.toString
                      () + ". Details: " + ex.getMessage());
1169          }
1170          catch (IOException ex) {
1171              errorReport.println("DELIVERY ERROR - Unable to communicate with host: "
                      + receiver.toString() + ". Details: " + ex.getMessage());
1172          }
1173      }
1174  }
1175
1176  public class JMixNetConfigManager extends ConfigManager {
1177
1178      protected static int DEFAULT\_PORT = 1981;
1179      protected static int DEFAULT\_MSG\_SIZE = 8192;
1180      protected static int DEFAULT\_MSG\_BATCH\_SIZE = 100;
1181      protected static int DEFAULT\_POOL\_SIZE = 50;
1182      protected static int DEFAULT\_FLUSH\_PERCENT = 70;
1183      protected static int DEFAULT\_FLUSH\_PERIOD = 500;
1184      protected static int DEFAULT\_MAX\_TRACK\_SIZE = 100;
1185
1186      protected int port;
```

```
1187    protected String address;
1188    protected int messageSize = DEFAULT\_MSG\_SIZE;
1189    protected int messageBatchSize = DEFAULT\_MSG\_BATCH\_SIZE;
1190    protected int poolSize = DEFAULT\_POOL\_SIZE;
1191    protected int flushPercent = DEFAULT\_FLUSH\_PERCENT;
1192    protected int flushPeriod = DEFAULT\_FLUSH\_PERIOD;
1193    protected int maxMsgTrackSize = DEFAULT\_MAX\_TRACK\_SIZE;
1194    protected String previousServer = null;
1195    protected String nextServer = null;
1196    protected int nextServerPort = DEFAULT\_PORT;
1197    protected String msgDelivererClass;
1198
1199    public JMixNetConfigManager() throws ConfigException {
1200        super("jmixnet.cfg");
1201        readConfiguration();
1202    }
1203
1204    public JMixNetConfigManager(InputStream config) throws ConfigException {
1205        super(config);
1206        readConfiguration();
1207    }
1208
1209    protected void readConfiguration() throws ConfigException {
1210        String chain;
1211
1212        //get the port to be used, default is DEFAULT\_PORT
1213        port = getInt("port", DEFAULT\_PORT);
1214
1215        //get the address to be used (dot-quadded or fully qualified), default is
              canonical hostname
1216        try {
1217            address = getProperty("address", InetAddress.getLocalHost().
                  getCanonicalHostName());
1218        }
1219        catch (UnknownHostException ex) {
1220            ex.printStackTrace();
1221        }
1222
1223        //get the message size to be used, default is DEFAUL\_MSG\_SIZE
1224        messageSize = getInt("messageSize", DEFAULT\_MSG\_SIZE);
1225        //get the message batch size to be used, default is DEFAULT\_MSG\_BATCH\_SIZE
1226        messageBatchSize = getInt("messageBatchSize", DEFAULT\_MSG\_BATCH\_SIZE);
1227        //get the message pool size to be used, default is DEFAULT\_POOL\_SIZE
1228        poolSize = getInt("poolSize", DEFAULT\_POOL\_SIZE);
```

```
1229        //get the flush percent to be used, default is DEFAULT\_FLUSH\_PERCENT
1230        flushPercent = getInt("flushPercent", DEFAULT\_FLUSH\_PERCENT);
1231        //get the flush period to be used, default is DEFAULT\_FLUSH\_PERIOD
1232        flushPeriod = getInt("flushPeriod", DEFAULT\_FLUSH\_PERIOD);
1233        //get the max message track size
1234        maxMsgTrackSize = getInt("maxMsgTrackSize", DEFAULT\_MAX\_TRACK\_SIZE);
1235        //get the message deliverer class, default is JMixNetDeliverer
1236        msgDelivererClass = getString("msgDelivererClass", "br.edu.ufsc.labsec.
                jmixnet.JMixNetDeliverer");
1237
1238        //get the chain to define the previous and next servers
1239        chain = getProperty("chain");
1240        if (chain == null) throw new MissingConfigException("chain");
1241
1242        //list all chain addresses
1243        String[] addresses = chain.split(",");
1244        String currentAddress = null;
1245        String[] addressPort = null;
1246
1247        //seek for local address
1248        for (int i = 0; i < addresses.length; i++) {
1249            //split the address part (before the colon) and the port part (after the
                    colon)
1250            addressPort = addresses[i].split(":");
1251
1252            if (//local address was found?
1253                (addressPort[0].equals(address))
1254                //config port is not set or is the same?
1255                (addressPort.length == 1 || (port == Integer.parseInt(addressPort[1])
                        )) )
1256            {
1257                //the previous server is the address of the previous iteration
1258                previousServer = currentAddress;
1259
1260                //verify if it is not the last
1261                if (i < addresses.length - 1) {
1262                    //set the next server info
1263                    addressPort = addresses[i + 1].split(":");
1264                    nextServer = addressPort[0];
1265                    if (addressPort.length > 1)
1266                        nextServerPort = Integer.parseInt(addressPort[1]);
1267                }
1268                break;
1269            }
```

```
1270                currentAddress = addressPort[0];
1271            }
1272            //verify that previous or next servers was set
1273            if ((previousServer == null)  (nextServer == null)) {
1274                throw new WrongConfigException("chain");
1275            }
1276        }
1277
1278        public String getAddress() {
1279            return address;
1280        }
1281
1282        public int getPort() {
1283            return port;
1284        }
1285
1286        public String getNextServer() {
1287            return nextServer;
1288        }
1289
1290        public int getNextServerPort() {
1291            return nextServerPort;
1292        }
1293
1294        public String getPreviousServer() {
1295            return previousServer;
1296        }
1297
1298        public int getMessageSize() {
1299            return messageSize;
1300        }
1301
1302        public int getMessageBatchSize() {
1303            return messageBatchSize;
1304        }
1305
1306        public int getPoolSize() {
1307            return poolSize;
1308        }
1309
1310        public int getFlushPercent() {
1311            return flushPercent;
1312        }
1313
```

```
1314        public int getFlushPeriod() {
1315            return flushPeriod;
1316        }
1317
1318        public String getMsgDelivererClass() {
1319            return msgDelivererClass;
1320        }
1321
1322        public String getKeyStoreLocation() {
1323            return getProperty("keyStoreLocation", "JMixNetServer.keystore");
1324        }
1325
1326        public String getKeyStorePassword() {
1327            return getProperty("keyStorePassword", "jmixnet");
1328        }
1329
1330        public Collection getServerCerts() throws ConfigException {
1331            try {
1332                return X509CertificateGenerator.genCertificates(
1333                    JMixNetConfigManager.class.getClassLoader().getResourceAsStream("
                        serverCertificates.b64"));
1334            }
1335            catch (CryptoException ex) {
1336                throw new ConfigException("Error generating certificates.", ex);
1337            }
1338        }
1339
1340        public int getMaxMsgTrackSize() {
1341            return maxMsgTrackSize;
1342        }
1343
1344        public HashMap getAllowedClients() {
1345            HashMap allowedSet = new HashMap();
1346            String property = getProperty("allowedClients");
1347            if (property != null) {
1348                String[] allowedClients = property.split(",");
1349                int numAllowed = allowedClients.length;
1350                for (int i = 0; i < numAllowed; i++) {
1351                    allowedSet.put(allowedClients[i], null);
1352                }
1353            }
1354            return allowedSet;
1355        }
1356    }
```

```
1357
1358 public class JMixNetClient {
1359
1360     //configuration-related
1361     protected JMixNetConfigManager config;
1362     protected Iterator iterator;
1363
1364     //message info
1365     protected int messageSize;
1366     protected int sessionKeyFieldSize;
1367     protected int dataDescSize = 2; //a short (2 bytes) describes the size of
          meaningful data
1368     protected final int sessionKeySize = 16;
1369     protected final int hashFieldSize = 16; //MD5
1370     protected byte[] encodedParameters;
1371
1372     //message building
1373     public int maxDataSize;
1374     protected ByteBuffer dataBuffer;
1375     protected ByteBuffer tempBuffer;
1376     protected byte[] srcBuf;
1377     protected byte[] dstBuf;
1378     protected int srcBufContentSize;
1379     protected int dstBufContentSize;
1380
1381     //reports
1382     protected PrintStream errorReport = System.err;
1383     protected PrintStream clientReport = System.out;
1384
1385     //cryptographers and hash
1386     protected SymCryptographer symCrypto;
1387     protected AsymCryptographer asymCrypto;
1388     protected MessageDigest digester;
1389
1390     //to receive and send messages
1391     protected ArrayList serverCerts;
1392     protected Socket connection;
1393     protected InputStream serverInput = null;
1394     protected OutputStream serverOutput = null;
1395
1396     /**
1397      * Construct the client with initial setup.
1398      */
1399     public JMixNetClient() throws JMixNetException {
```

```
1400            try {
1401                config = new JMixNetConfigManager();
1402            }
1403            catch (ConfigException ex) {
1404                showError("Config error occurred:" + ex.getMessage());
1405                throw new JMixNetException();
1406            }
1407            configureClient();
1408        }
1409
1410        /**
1411         * Construct the client with initial setup, based on specified config file.
1412         *
1413         * @param configFile Configuration file other than "jmixnet.cfg"
1414         */
1415        public JMixNetClient(InputStream configStream) throws JMixNetException {
1416            try {
1417                config = new JMixNetConfigManager(configStream);
1418            }
1419            catch (ConfigException ex) {
1420                showError("Config error occurred:" + ex.getMessage());
1421                throw new JMixNetException();
1422            }
1423            configureClient();
1424        }
1425
1426        /**
1427         * Configurations based on config file.
1428         */
1429        protected void configureClient() throws JMixNetException {
1430
1431            //ensure Bouncy Castle Provider is available
1432            if (Security.getProvider("BC") == null) {
1433                Security.addProvider(new BouncyCastleProvider());
1434            }
1435
1436            //message config
1437            messageSize = config.getMessageSize();
1438            dataBuffer = ByteBuffer.allocate(messageSize);
1439            tempBuffer = ByteBuffer.allocate(messageSize);
1440
1441            try {
1442                //server certificates
1443                serverCerts = (ArrayList)config.getServerCerts();
```

```
1444            Collections.reverse(serverCerts);
1445        }
1446        catch (ConfigException ex) {
1447            showError("Unable to get server certificates:" + ex.getMessage());
1448            throw new JMixNetException(ex);
1449        }
1450        //calculate max data size the client can send
1451        iterator = serverCerts.iterator();
1452        int totalKeyLength = 0;
1453        while (iterator.hasNext()) {
1454            totalKeyLength += X509CertificateInfo.getKeyBytesSize((X509Certificate)
                    iterator.next());
1455        }
1456        maxDataSize = messageSize - hashFieldSize - totalKeyLength - dataDescSize;
1457
1458        try {
1459            //cryptography services and hash
1460            symCrypto = new SymCryptographer();
1461            asymCrypto = new AsymCryptographer();
1462            digester = MessageDigest.getInstance("MD5");
1463        }
1464        catch (CryptoException ex) {
1465            showError("Unable to use cryptography services:" + ex.getMessage());
1466            throw new JMixNetException();
1467        }
1468        catch (NoSuchAlgorithmException ex) {
1469            showError("Unable to get a message digest instance:" + ex.getMessage());
1470            throw new JMixNetException();
1471        }
1472    }
1473
1474    /**
1475     * Uses error report to display desired error message.
1476     *
1477     * @param message Error message to be shown.
1478     */
1479    protected void showError(String message) {
1480        errorReport.println("ERROR: " + message);
1481    }
1482
1483    /**
1484     * Uses client report to display desired log message.
1485     *
1486     * @param message Log message to be shown.
```

```
1487         */
1488        protected void showLog(String message) {
1489            clientReport.println("Log: " + message);
1490        }
1491
1492        /**
1493         * Set the PrintStream object to be used as the error report.
1494         *
1495         * @param report PrintStream object to be used as the error report.
1496         */
1497        public void setErrorReport(PrintStream report) {
1498            errorReport = report;
1499        }
1500
1501        /**
1502         * Set the PrintStream object to be used as the client report.
1503         *
1504         * @param report PrintStream object to be used as the server report.
1505         */
1506        public void setClientReport(PrintStream report) {
1507            clientReport = report;
1508        }
1509
1510        /**
1511         * Connect to JMixNet server using configuration data.
1512         */
1513        public void connect() {
1514            try {
1515                connection = new Socket(config.getAddress(), config.getPort());
1516                serverInput = connection.getInputStream();
1517                serverOutput = connection.getOutputStream();
1518            }
1519            catch (UnknownHostException ex) {
1520                showError("Unknown host: " + connection.toString() + ".Details: " + ex.
                        getMessage());
1521            }
1522            catch (IOException ex) {
1523                showError("Unable to communicate with JMixNet server.Details: " + ex.
                        getMessage());
1524            }
1525        }
1526
1527        /**
1528         * Disconnect from JMixNet server.
```

```
1529         */
1530        public void disconnect() {
1531            try {
1532                connection.close();
1533            }
1534            catch (IOException ex) {
1535                showError("Unable to disconnect from JMixNet server.Details: " + ex.
                        getMessage());
1536            }
1537        }
1538
1539        /**
1540         * Send desired data to JMixNet server.
1541         *
1542         * @param data Data to be sent.
1543         * @throws JMixNetException If data lenght is too long.
1544         */
1545        public void sendData(byte[] data, int dataLength) throws JMixNetException {
1546            //verify data length
1547            if (dataLength > maxDataSize) {
1548                throw new DataLengthException("Data length exceeded, maximum allowed is "
                        + maxDataSize + " bytes.");
1549            }
1550            //put the address, port, meaningful data size, and meaningful data
1551            tempBuffer.clear();
1552            tempBuffer.putShort((short)dataLength);
1553            tempBuffer.put(data, 0, dataLength);
1554
1555            //verify the need for padding
1556            int padding = ((dataDescSize + dataLength) % sessionKeySize > 0) ?
1557                sessionKeySize - ((dataDescSize + dataLength) % sessionKeySize) : 0;
1558            srcBufContentSize = dataDescSize + dataLength + padding;
1559
1560            try {
1561                int i;
1562                //take each certificate and build the digital envelopes
1563                for (i = 0; i < serverCerts.size(); i++) {
1564                    //define buffers roles
1565                    srcBuf = (i % 2 == 0) ? tempBuffer.array() : dataBuffer.array();
1566                    dstBuf = (i % 2 == 0) ? dataBuffer.array() : tempBuffer.array();
1567
1568                    //set the server certificate
1569                    asymCrypto.setCertificate((Certificate)serverCerts.get(i));
1570                    sessionKeyFieldSize = X509CertificateInfo.getKeyBytesSize((
```

```
                X509Certificate)serverCerts.get(i));
1571
1572            //set symmetric key and get encoded parameters
1573            symCrypto.generateKey();
1574
1575            //build n-1 envelopes
1576            if (i < serverCerts.size() - 1) {
1577                //verify if this is the one before the last
1578                if (i < serverCerts.size() - 2) {
1579                    //encrypt the source
1580                    symCrypto.encrypt(srcBuf, 0, srcBufContentSize, dstBuf,
                            sessionKeyFieldSize);
1581                    srcBufContentSize += sessionKeyFieldSize;
1582
1583                    //encrypt simmetric key and cipher parameters
1584                    asymCrypto.addDataToEncrypt(symCrypto.getEncodedKey(), 0,
                            symCrypto.getEncodedKey().length, dstBuf, 0);
1585                    encodedParameters = symCrypto.getParameters().getEncoded();
1586                    asymCrypto.encrypt(encodedParameters, 0, encodedParameters.
                            length, dstBuf);
1587                }
1588                //the one before the last needs to take into account the hash
                        code to the last envelope
1589                else {
1590                    //encrypt the source
1591                    symCrypto.encrypt(srcBuf, 0, srcBufContentSize, dstBuf,
                            sessionKeyFieldSize + hashFieldSize);
1592
1593                    //encrypt simmetric key
1594                    asymCrypto.addDataToEncrypt(symCrypto.getEncodedKey(), 0,
                            symCrypto.getEncodedKey().length, dstBuf, 0);
1595                    encodedParameters = symCrypto.getParameters().getEncoded();
1596                    asymCrypto.encrypt(encodedParameters, 0, encodedParameters.
                            length, dstBuf, hashFieldSize);
1597                }
1598            }
1599            //build the last envelope
1600            else {
1601                //calculate the hash of the source (with padding) and add it to
                        the begining of the source
1602                digester.update(srcBuf, hashFieldSize, messageSize -
                        sessionKeyFieldSize - hashFieldSize);
1603                digester.digest(srcBuf, 0, hashFieldSize);
1604
```

```
1605                    //encrypt the source
1606                    symCrypto.encrypt(srcBuf, 0, messageSize - sessionKeyFieldSize,
                            dstBuf, sessionKeyFieldSize);
1607
1608                    //encrypt simmetric key
1609                    asymCrypto.addDataToEncrypt(symCrypto.getEncodedKey(), 0,
                            symCrypto.getEncodedKey().length, dstBuf, 0);
1610                    encodedParameters = symCrypto.getParameters().getEncoded();
1611                    asymCrypto.encrypt(encodedParameters, 0, encodedParameters.length
                            , dstBuf);
1612                }
1613                if (encodedParameters.length != 18) System.err.println("Parameter
                        Length diff from 18");
1614            }
1615            //send the message
1616            serverOutput.write(dstBuf);
1617        }
1618        catch (Exception ex) {
1619            throw new JMixNetException(ex);
1620        }
1621    }
1622
1623    /**
1624     * When running the client directly, with no application tayloring, this main
              method is used.
1625     *
1626     * @param args Config file name, when "jmixnet.cfg" is not supposed to be used.
1627     * @throws JMixNetException On unrecoverable error.
1628     */
1629    public static void main(String[] args) throws JMixNetException,
            FileNotFoundException {
1630
1631        JMixNetClient client;
1632        ByteBuffer buffer = ByteBuffer.allocate(1024);
1633
1634        //to read data from console
1635        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
1636        String line;
1637
1638        //send messages to example server (br.edu.ufsc.labsec.jmixnet.example.
              ReceiverServer)
1639        InetAddress receiverAddress = null;
1640        short receiverPort = 2000;
1641        try {
```

```
1642            //receiverAddress = InetAddress.getLocalHost();
1643            receiverAddress = InetAddress.getByName("192.168.128.11");
1644        }
1645        catch (UnknownHostException ex) {
1646            ex.printStackTrace();
1647        }
1648        //start client with desired config file
1649        if (0 == args.length)
1650            client = new JMixNetClient();
1651        else
1652            client = new JMixNetClient(new FileInputStream(args[0]));
1653
1654        System.out.println("Client running.");
1655        client.connect();
1656
1657        //send lines read from console until "exit" is typed
1658        /*line = reader.readLine();
1659        while (!line.equals("exit")) {
1660            client.sendData((line + "").getBytes());
1661            line = reader.readLine();
1662        }*/
1663
1664        long lastTime = new Date().getTime();
1665        long now;
1666        double elapsedTime;
1667        int sentMsgs = 0;
1668        String numero;
1669        int maxDataSize = client.maxDataSize;
1670        byte[] buf = new byte[maxDataSize];
1671        int totalMsgs = 2000;
1672        int i = 0;
1673        //for (i = 0; i < totalMsgs; i++) {
1674        do {
1675            //verify exit condition
1676            try {i = System.in.available();} catch (IOException ex) {}
1677
1678            //send message
1679            numero = (new Integer(i)).toString();
1680            buffer.clear();
1681            buffer.put(receiverAddress.getAddress());
1682            buffer.putShort(receiverPort);
1683            buffer.put(numero.getBytes());
1684            client.sendData(buffer.array(), numero.getBytes().length + 6);
1685            //client.sendData(buf, maxDataSize);
```

```
1686                sentMsgs++;
1687
1688                //calculate throughput
1689                now = new Date().getTime();
1690                elapsedTime = (now - lastTime) / 1000.0;
1691                if (elapsedTime > 5) {
1692                    lastTime = now;
1693                    System.out.println("Speed: " + sentMsgs / elapsedTime + " msg/s.");
1694                    sentMsgs = 0;
1695                }
1696            } while (i == 0);
1697            client.disconnect();
1698        }
1699    }
1700
1701    public class DelivererThread extends Thread {
1702
1703        //message delivery
1704        protected LinkedList newMessages;
1705        protected LinkedList workingMessages;
1706        protected LinkedList handledMessages;
1707        protected MessageDeliverer deliverer = null;
1708        protected OutputStream serverOutput;
1709        protected boolean deliveryActive = true;
1710        protected Iterator it;
1711        protected ByteBuffer currentBuffer;
1712
1713        /**
1714         * Creates a new deliverer thread to be used by the last server.
1715         *
1716         * @param msgDelivererClass The Class to be used to get a deliverer object.
1717         * @throws JMixNetException If the deliverer cannot be created.
1718         */
1719        public DelivererThread(String msgDelivererClass) throws JMixNetException {
1720            try {
1721                //creates the message deliverer
1722                deliverer = (MessageDeliverer)(Class.forName(msgDelivererClass)).
                        newInstance();
1723            }
1724            catch (InstantiationException ex) {
1725                throw new JMixNetException("Unable to instantiate message deliverer.
                        Details: " + ex.getMessage());
1726            }
1727            catch (IllegalAccessException ex) {
```

```
1728                throw new JMixNetException("Unable to instantiate message deliverer.
                        Details: " + ex.getMessage());
1729            }
1730            catch (ClassNotFoundException ex) {
1731                throw new JMixNetException("Unable to instantiate message deliverer.
                        Details: " + ex.getMessage());
1732            }
1733            setupBuffers();
1734        }
1735
1736        /**
1737         * Creates a new deliverer thread to be used by a middle server.
1738         */
1739        public DelivererThread() {
1740            setupBuffers();
1741        }
1742
1743        /**
1744         * Creates needed buffers.
1745         */
1746        protected void setupBuffers() {
1747            newMessages = new LinkedList();
1748            handledMessages = new LinkedList();
1749            workingMessages = new LinkedList();
1750        }
1751
1752        /**
1753         * Add new messages to be handled, and return the buffers already handled.
1754         *
1755         * @param newMsgs New messages to be delivered.
1756         * @return The buffers already handled.
1757         */
1758        public LinkedList exchangeBuffers(LinkedList newMsgs) {
1759
1760            //add received new messages
1761            synchronized(newMessages) {
1762                newMessages.addAll(newMsgs);
1763            }
1764            newMsgs.clear();
1765
1766            //return handled messages
1767            synchronized(handledMessages) {
1768                newMsgs.addAll(handledMessages);
1769                handledMessages.clear();
```

```
1770            }
1771            //tell the thread that there are new messages
1772            interrupt();
1773
1774            return newMsgs;
1775        }
1776
1777        /**
1778         * Tells the thread to stop working.
1779         */
1780        public void finishDelivery() {
1781            deliveryActive = false;
1782            interrupt();
1783        }
1784
1785        /**
1786         * Sets the server output to be used to deliver messages.
1787         *
1788         * @param serverOutput The serverOutput to set.
1789         */
1790        public void setServerOutput(OutputStream serverOutput) {
1791            this.serverOutput = serverOutput;
1792        }
1793
1794        public void run() {
1795
1796            while (deliveryActive) {
1797                if (newMessages.size() > 0) {
1798                    //get the new messages
1799                    synchronized(newMessages) {
1800                        workingMessages.addAll(newMessages);
1801                        newMessages.clear();
1802                    }
1803                    it = workingMessages.iterator();
1804                    while (it.hasNext()) {
1805                        currentBuffer = (ByteBuffer)it.next();
1806                        if (deliverer != null)
1807                            deliverer.deliverMsg(currentBuffer);
1808                        else
1809                            try {
1810                                serverOutput.write(currentBuffer.array());
1811                            }
1812                            catch (IOException ex) {
1813                                System.err.println("Unable to deliver message to the next
```

```
                                server.");
1814                        }
1815                    currentBuffer.clear();
1816                    handledMessages.add(currentBuffer);
1817                }
1818                workingMessages.clear();
1819            }
1820            else {
1821                try {
1822                    //wait a second to verify new messages
1823                    sleep(1000);
1824                }
1825                catch (InterruptedException e) {
1826                    //do nothing
1827                }
1828            }
1829        }
1830    }
1831 }
1832
1833 public class DataLengthException extends JMixNetException {
1834
1835     public DataLengthException(String msg) {
1836         super(msg);
1837     }
1838
1839     public DataLengthException(String msg, Throwable cause) {
1840         super(msg, cause);
1841     }
1842 }
1843
1844
1845 public class X509CertificateInfo {
1846
1847     public static int getKeyBytesSize(X509Certificate cert) {
1848         int keyLength = cert.getPublicKey().getEncoded().length;
1849         return keyLength - (keyLength % 64);
1850     }
1851 }
1852
1853 public class X509CertificateGenerator {
1854
1855     protected static CertificateFactory factory = null;
1856
```

```
1857    /**
1858     * Get the internal X509 factory, instantiating it if it's not available yet.
1859     *
1860     * @return The internal X509 factory.
1861     */
1862    protected static CertificateFactory getFactory() {
1863        if (factory == null) {
1864            try {
1865                factory = CertificateFactory.getInstance("X509");
1866            }
1867            catch (CertificateException e) {
1868                System.err.println("Error creating CertificateFactory.");
1869                e.printStackTrace();
1870            }
1871        }
1872        return factory;
1873    }
1874
1875    /**
1876     * Generate a collection containing the certificates extracted from the specified
              file.
1877     *
1878     * @param fileName The file containing the certificates.
1879     * @return A certificate collection.
1880     * @throws CryptoException If some parse error ocurred.
1881     */
1882    public static Collection genCertificates(String fileName) throws CryptoException
          {
1883        try {
1884            return getFactory().generateCertificates(new FileInputStream(fileName));
1885        }
1886        catch (CertificateException ex) {
1887            throw new CryptoException("Certificate parse error during certificate
                  generation.", ex);
1888        }
1889        catch (FileNotFoundException ex) {
1890            throw new CryptoException("Certificates file not found.", ex);
1891        }
1892    }
1893
1894    /**
1895     * Generate a collection containing the certificates extracted from the specified
              stream.
1896     *
```

```
1897        * @param certStream The stream containing the certificates.
1898        * @return A certificate collection.
1899        * @throws CryptoException If some parse error ocurred.
1900        */
1901       public static Collection genCertificates(InputStream certStream) throws
               CryptoException {
1902           try {
1903               return getFactory().generateCertificates(certStream);
1904           }
1905           catch (CertificateException ex) {
1906               throw new CryptoException("Certificate parse error during certificate
                       generation.", ex);
1907           }
1908       }
1909   }
1910
1911
1912   public class SymCryptographer {
1913
1914       protected Cipher cipher;
1915       protected String algorithm = "AES";
1916       protected String transformation = "AES/CBC/NoPadding";
1917
1918       protected KeyGenerator keyGenerator;
1919       protected SecretKey symKey;
1920       protected AlgorithmParameters parameters;
1921       protected boolean hasParameters = false;
1922
1923       protected static final int OP\_NONE    = 0;
1924       protected static final int OP\_ENCRYPT = 1;
1925       protected static final int OP\_DECRYPT = 2;
1926       protected static final int OP\_SIGN    = 3;
1927       protected static final int OP\_VERIFY  = 4;
1928       protected int currentOperation = OP\_NONE;
1929
1930       /**
1931        * Constructs a new symmetric cryptographer.
1932        */
1933       public SymCryptographer() throws CryptoException {
1934           try {
1935               cipher = Cipher.getInstance(transformation);
1936               keyGenerator = KeyGenerator.getInstance(algorithm);
1937           }
1938           catch (NoSuchAlgorithmException ex) {
```

```
1939            throw new CryptoException("Error with algorithm definitions.", ex);
1940        }
1941        catch (NoSuchPaddingException ex) {
1942            throw new CryptoException("Error with padding definitions.", ex);
1943        }
1944    }
1945
1946    /**
1947     * Generates a new ramdom key.
1948     */
1949    public void generateKey() {
1950        symKey = keyGenerator.generateKey();
1951        currentOperation = OP\_NONE;
1952    }
1953
1954    /**
1955     * Gets the key material of the symmetric key.
1956     *
1957     * @return The key material of the symmetric key.
1958     */
1959    public byte[] getEncodedKey() {
1960        return symKey.getEncoded();
1961    }
1962
1963    /**
1964     * Sets the key material of the symmetric key.
1965     *
1966     * @param key key material.
1967     * @param offset offset of <tt>key</tt> where key material starts.
1968     * @param len key material length.
1969     */
1970    public void setEncodedKey(byte[] key, int offset, int len) {
1971        symKey = new SecretKeySpec(key, offset, len, algorithm);
1972        currentOperation = OP\_NONE;
1973    }
1974
1975    /**
1976     * Set the cipher parameters to be used.
1977     *
1978     * @param params The encoded parameters.
1979     * @param offset The offset in <tt>params</tt> where they start.
1980     * @param len The parameters length.
1981     * @throws CryptoException If a parameter initialization error occurred.
1982     */
```

```
1983    public void setEncodedParameters(byte[] params, int offset, int len) throws
            CryptoException {
1984        try {
1985            byte[] newParams = new byte[len];
1986            System.arraycopy(params, offset, newParams, 0, len);
1987            parameters = AlgorithmParameters.getInstance(algorithm);
1988            parameters.init(newParams);
1989            hasParameters = true;
1990            currentOperation = OP\_NONE;
1991        }
1992        catch (IOException ex) {
1993            throw new CryptoException("Erro with parameter initialization.", ex);
1994        }
1995        catch (NoSuchAlgorithmException ex) {
1996            throw new CryptoException("Error with algorithm definitions.", ex);
1997        }
1998    }
1999
2000    /**
2001     * Set the cipher parameters to be used.
2002     *
2003     * @param params The encoded parameters.
2004     * @throws CryptoException If a parameter initialization error occurred.
2005     */
2006    public void setEncodedParameters(byte[] params) throws CryptoException {
2007        try {
2008            parameters = AlgorithmParameters.getInstance(algorithm);
2009            parameters.init(params);
2010            hasParameters = true;
2011            currentOperation = OP\_NONE;
2012        }
2013        catch (IOException ex) {
2014            throw new CryptoException("Erro with parameter initialization.", ex);
2015        }
2016        catch (NoSuchAlgorithmException ex) {
2017            throw new CryptoException("Error with algorithm definitions.", ex);
2018        }
2019    }
2020
2021    /**
2022     * Discard the cipher parameters in use.
2023     */
2024    public void resetParameters() {
2025        hasParameters = false;
```

```
2026        }
2027
2028        /**
2029         * Returns the parameters used with this cipher.
2030         *
2031         * @return the parameters used with this cipher, or null if this cipher does not
                 use any parameters.
2032         */
2033        public AlgorithmParameters getParameters() {
2034            return cipher.getParameters();
2035        }
2036
2037        /**
2038         * Verify if the cipher is initialized with the desired operation.
2039         *
2040         * @param operation Desired operation
2041         * @throws CryptoException If an invalid key or algorithm parameter is found.
2042         */
2043        protected void verifyOperation(int operation) throws CryptoException {
2044            try {
2045                switch (operation) {
2046                    case OP\_ENCRYPT :
2047                        if (currentOperation != OP\_ENCRYPT) {
2048                            currentOperation = OP\_ENCRYPT;
2049                            if (hasParameters)
2050                                cipher.init(Cipher.ENCRYPT\_MODE, symKey, parameters);
2051                            else
2052                                cipher.init(Cipher.ENCRYPT\_MODE, symKey);
2053                        }
2054                        break;
2055                    case OP\_DECRYPT :
2056                        if (currentOperation != OP\_DECRYPT) {
2057                            currentOperation = OP\_DECRYPT;
2058                            if (hasParameters)
2059                                cipher.init(Cipher.DECRYPT\_MODE, symKey, parameters);
2060                            else
2061                                cipher.init(Cipher.DECRYPT\_MODE, symKey);
2062                        }
2063                        break;
2064                }
2065            }
2066            catch (InvalidKeyException ex) {
2067                throw new CryptoException("Invalid symmetric key.", ex);
2068            }
```

```
2069            catch (InvalidAlgorithmParameterException ex) {
2070                throw new CryptoException("Invalid algorithm parameter.", ex);
2071            }
2072        }
2073
2074        /**
2075         * Returns the desired plain data encrypted.
2076         *
2077         * @param plainData the data to be encrypted.
2078         * @param inputOffset
2079         * @param inputLen
2080         * @param output
2081         * @param outputOffset
2082         * @return the encrypted data.
2083         * @throws CryptoException if some error occurred.
2084         */
2085        public int encrypt(byte[] plainData, int inputOffset, int inputLen, byte[] output
                , int outputOffset) throws CryptoException {
2086            try {
2087                verifyOperation(OP\_ENCRYPT);
2088                return cipher.doFinal(plainData, inputOffset, inputLen, output,
                        outputOffset);
2089            }
2090            catch (BadPaddingException ex) {
2091                throw new CryptoException("Erro with data padding.", ex);
2092            }
2093            catch (IllegalBlockSizeException ex) {
2094                throw new CryptoException("Erro with data block size.", ex);
2095            }
2096            catch (IllegalStateException ex) {
2097                throw new CryptoException("Wrong state found.", ex);
2098            }
2099            catch (ShortBufferException ex) {
2100                throw new CryptoException("Buffer too small.", ex);
2101            }
2102        }
2103
2104        /**
2105         * Returns the desired plain data encrypted.
2106         *
2107         * @param plainData the data to be encrypted.
2108         * @return the encrypted data.
2109         * @throws CryptoException if a padding or block size error occurred.
2110         */
```

```
2111    public byte[] encrypt(byte[] plainData) throws CryptoException {
2112        try {
2113            verifyOperation(OP\_ENCRYPT);
2114            return cipher.doFinal(plainData);
2115        }
2116        catch (BadPaddingException ex) {
2117            throw new CryptoException("Erro with data padding.", ex);
2118        }
2119        catch (IllegalBlockSizeException ex) {
2120            throw new CryptoException("Erro with data block size.", ex);
2121        }
2122    }
2123
2124    /**
2125     * Returns the desired encrypted data decrypted.
2126     *
2127     * @param encryptedData the data to be decrypted.
2128     * @return the decrypted data.
2129     * @throws CryptoException if a padding or block size error occurred.
2130     */
2131    public byte[] decrypt(byte[] encryptedData) throws CryptoException {
2132        try {
2133            verifyOperation(OP\_DECRYPT);
2134            return cipher.doFinal(encryptedData);
2135        }
2136        catch (BadPaddingException ex) {
2137            throw new CryptoException("Erro with data padding.", ex);
2138        }
2139        catch (IllegalBlockSizeException ex) {
2140            throw new CryptoException("Erro with data block size.", ex);
2141        }
2142    }
2143
2144    /**
2145     * Returns the desired encrypted data decrypted.
2146     *
2147     * @param encryptedData the data to be decrypted.
2148     * @param offset the offset in <tt>encryptedData</tt> where the encrypted data
            starts.
2149     * @param length the encrypted data length.
2150     * @return the decrypted data.
2151     * @throws CryptoException if a padding or block size error occurred.
2152     */
2153    public byte[] decrypt(byte[] encryptedData, int offset, int length) throws
```

```
        CryptoException {
2154        try {
2155            verifyOperation(OP\_DECRYPT);
2156            return cipher.doFinal(encryptedData, offset, length);
2157        }
2158        catch (BadPaddingException ex) {
2159            throw new CryptoException("Erro with data padding.", ex);
2160        }
2161        catch (IllegalBlockSizeException ex) {
2162            throw new CryptoException("Erro with data block size.", ex);
2163        }
2164    }
2165
2166    /**
2167     * Returns the desired encrypted data decrypted.
2168     *
2169     * @param encryptedData the data to be decrypted.
2170     * @param offset the offset in <tt>encryptedData</tt> where the encrypted data
            starts.
2171     * @param length the encrypted data length.
2172     * @param output the byte buffer to write the decrypted data.
2173     * @return the amount of decrypted data.
2174     * @throws CryptoException if a padding or block size error occurred.
2175     */
2176    public int decrypt(byte[] encryptedData, int offset, int length, byte[] output)
        throws CryptoException {
2177        try {
2178            verifyOperation(OP\_DECRYPT);
2179            return cipher.doFinal(encryptedData, offset, length, output);
2180        }
2181        catch (BadPaddingException ex) {
2182            throw new CryptoException("Erro with data padding.", ex);
2183        }
2184        catch (IllegalBlockSizeException ex) {
2185            throw new CryptoException("Erro with data block size.", ex);
2186        }
2187        catch (IllegalStateException ex) {
2188            throw new CryptoException("Wrong state found.", ex);
2189        }
2190        catch (ShortBufferException ex) {
2191            throw new CryptoException("Buffer too small.", ex);
2192        }
2193    }
2194
```

```
2195        /**
2196         * @return The transformation in use.
2197         */
2198        public String getTransformation() {
2199            return transformation;
2200        }
2201
2202        /**
2203         * @param string The transformation to be used.
2204         */
2205        public void setTransformation(String string) throws NoSuchPaddingException,
                NoSuchAlgorithmException {
2206            transformation = string;
2207
2208            //initialize cipher and key generator
2209            cipher = Cipher.getInstance(transformation);
2210            currentOperation = OP\_NONE;
2211            algorithm = transformation.split("/")[0];
2212            keyGenerator = KeyGenerator.getInstance(algorithm);
2213        }
2214    }
2215
2216
2217    public class CryptoException extends Exception {
2218
2219        public CryptoException(String msg) {
2220            super(msg);
2221        }
2222
2223        public CryptoException(String msg, Throwable cause) {
2224            super(msg, cause);
2225        }
2226    }
2227
2228    public class AsymCryptographer {
2229
2230        protected KeyStore keyStore;
2231        protected char[] keyStorePassword;
2232        protected Cipher cipher;
2233        protected Key privateKey = null;
2234        protected Certificate certificate = null;
2235
2236        protected static final int OP\_NONE    = 0;
2237        protected static final int OP\_ENCRYPT = 1;
```

```
2238        protected static final int OP\_DECRYPT = 2;
2239        protected static final int OP\_SIGN    = 3;
2240        protected static final int OP\_VERIFY  = 4;
2241        protected int currentOperation = OP\_NONE;
2242
2243        protected String transformation = "RSA";
2244
2245        /**
2246         * Constructs a new asymmetric cryptographer using the keystore information
                provided.
2247         *
2248         * @param keyStoreFilename Name of the file containing the keystore.
2249         * @param keyStorePassword Password to open the keystore.
2250         *
2251         * @throws CryptoException if some unexpected error occurred.
2252         */
2253        public AsymCryptographer(String keyStoreFilename, String keyStorePass) throws
              CryptoException {
2254            try {
2255                keyStore = KeyStore.getInstance("JKS");
2256                keyStorePassword = keyStorePass.toCharArray();
2257                keyStore.load(new FileInputStream(keyStoreFilename), keyStorePassword);
2258
2259                cipher = Cipher.getInstance(transformation);
2260            }
2261            catch (KeyStoreException ex) {
2262                throw new CryptoException("Error opening keystore file.", ex);
2263            }
2264            catch (IOException ex) {
2265                throw new CryptoException("Error reading keystore file.", ex);
2266            }
2267            catch (NoSuchAlgorithmException ex) {
2268                throw new CryptoException("Error with algorithm definitions.", ex);
2269            }
2270            catch (CertificateException ex) {
2271                throw new CryptoException("Error loading certificate from keystore.", ex)
                      ;
2272            }
2273            catch (NoSuchPaddingException ex) {
2274                throw new CryptoException("Error with padding definitions.", ex);
2275            }
2276        }
2277
2278        /**
```

```
2279        * Constructs a new asymmetric cryptographer using the keystore information
                provided.
2280        *
2281        * @throws CryptoException if some unexpected error occurred.
2282        */
2283       public AsymCryptographer() throws CryptoException {
2284           try {
2285               cipher = Cipher.getInstance(transformation);
2286           }
2287           catch (NoSuchAlgorithmException ex) {
2288               throw new CryptoException("Error with algorithm definitions.", ex);
2289           }
2290           catch (NoSuchPaddingException ex) {
2291               throw new CryptoException("Error with padding definitions.", ex);
2292           }
2293       }
2294
2295       /**
2296        * Tells the criptographer to use the desired alias content.
2297        *
2298        * @param newAlias The alias to be used hereafter.
2299        * @param password The alias password. Null to use the keystore password.
2300        * @throws CryptoException If some keystore error occurred, or a bad password was
                informed.
2301        */
2302       public void useAlias(String newAlias, String password) throws CryptoException {
2303           try {
2304               //verify and use the private key contained in alias
2305               if (keyStore.isKeyEntry(newAlias)) {
2306                   if (password == null) {
2307                       privateKey = keyStore.getKey(newAlias, keyStorePassword);
2308                   }
2309                   else {
2310                       privateKey = keyStore.getKey(newAlias, password.toCharArray());
2311                   }
2312               }
2313               //use the certificate contained in alias
2314               certificate = keyStore.getCertificate(newAlias);
2315               currentOperation = OP\_NONE;
2316           }
2317           catch (KeyStoreException ex) {
2318               throw new CryptoException("Error using alias.", ex);
2319           }
2320           catch (NoSuchAlgorithmException ex) {
```

```
2321            throw new CryptoException("Error using alias.", ex);
2322        }
2323        catch (UnrecoverableKeyException ex) {
2324            throw new CryptoException("Error using alias, maybe password is wrong.",
                    ex);
2325        }
2326    }
2327
2328    /**
2329     * Sets the certificate to be used.
2330     *
2331     * @param cert The certificate to be used.
2332     */
2333    public void setCertificate(Certificate cert) {
2334        certificate = cert;
2335        currentOperation = OP\_NONE;
2336    }
2337
2338    /**
2339     * Gets the certificate in use.
2340     *
2341     * @return The certificate in use.
2342     */
2343    public Certificate getCertificate() {
2344        return certificate;
2345    }
2346
2347    /**
2348     * Verify if the cipher is initialized with the desired operation.
2349     *
2350     * @param operation Desired operation
2351     * @throws CryptoException If an invalid key or algorithm parameter is found.
2352     */
2353    protected void verifyOperation(int operation) throws CryptoException {
2354        try {
2355            switch (operation) {
2356                case OP\_ENCRYPT :
2357                    if (currentOperation != OP\_ENCRYPT) {
2358                        currentOperation = OP\_ENCRYPT;
2359                        cipher.init(Cipher.ENCRYPT\_MODE, certificate);
2360                    }
2361                    break;
2362                case OP\_DECRYPT :
2363                    if (currentOperation != OP\_DECRYPT) {
```

```
2364                        currentOperation = OP\_DECRYPT;
2365                        cipher.init(Cipher.DECRYPT\_MODE, privateKey);
2366                    }
2367                    break;
2368                }
2369            }
2370            catch (InvalidKeyException ex) {
2371                throw new CryptoException("Invalid symmetric key.", ex);
2372            }
2373        }
2374
2375        /**
2376         * Returns the desired plain data encrypted using the certificate defined
                previously.
2377         *
2378         * @param plainData the data to be encrypted.
2379         * @return the encrypted data.
2380         * @throws CryptoException if a padding or block size error occurred.
2381         */
2382        public byte[] encrypt(byte[] plainData) throws CryptoException {
2383            try {
2384                verifyOperation(OP\_ENCRYPT);
2385                return cipher.doFinal(plainData);
2386            }
2387            catch (BadPaddingException ex) {
2388                throw new CryptoException("Error with data padding.", ex);
2389            }
2390            catch (IllegalBlockSizeException ex) {
2391                throw new CryptoException("Error with data block size.", ex);
2392            }
2393        }
2394
2395        /**
2396         * Returns the desired plain data encrypted using the certificate defined
                previously.
2397         *
2398         * @param plainData the data to be encrypted.
2399         * @param offset the offset in <tt>plainData</tt> where the plain data starts.
2400         * @param length the plain data length.
2401         * @param output the byte buffer to write the encrypted data.
2402         * @return the amount of encrypted data.
2403         * @throws CryptoException if some error occurred.
2404         */
2405        public int encrypt(byte[] plainData, int offset, int length, byte[] output)
```

```
        throws CryptoException {
2406        try {
2407            verifyOperation(OP\_ENCRYPT);
2408            return cipher.doFinal(plainData, offset, length, output);
2409        }
2410        catch (BadPaddingException ex) {
2411            throw new CryptoException("Error with data padding.", ex);
2412        }
2413        catch (IllegalBlockSizeException ex) {
2414            throw new CryptoException("Error with data block size.", ex);
2415        }
2416        catch (ShortBufferException ex) {
2417            throw new CryptoException("Buffer too small.", ex);
2418        }
2419    }
2420
2421    /**
2422     * Returns the desired plain data encrypted using the certificate defined
            previously.
2423     *
2424     * @param plainData the data to be encrypted.
2425     * @param offset the offset in <tt>plainData</tt> where the plain data starts.
2426     * @param length the plain data length.
2427     * @param output the byte buffer to write the encrypted data.
2428     * @param outputOffset the offset in <tt>output</tt> where the encrypted data
            starts.
2429     * @return the amount of encrypted data.
2430     * @throws CryptoException if some error occurred.
2431     */
2432    public int encrypt(byte[] plainData, int offset, int length, byte[] output, int
        outputOffset) throws CryptoException {
2433        try {
2434            verifyOperation(OP\_ENCRYPT);
2435            return cipher.doFinal(plainData, offset, length, output, outputOffset);
2436        }
2437        catch (BadPaddingException ex) {
2438            throw new CryptoException("Error with data padding.", ex);
2439        }
2440        catch (IllegalBlockSizeException ex) {
2441            throw new CryptoException("Error with data block size.", ex);
2442        }
2443        catch (ShortBufferException ex) {
2444            throw new CryptoException("Buffer too small.", ex);
2445        }
```

```
2446        }
2447
2448        /**
2449         * Returns the desired encrypted data decrypted using the private key defined
                previously.
2450         *
2451         * @param encryptedData the data to be decrypted.
2452         * @return the decrypted data.
2453         * @throws CryptoException if a padding or block size error occurred.
2454         */
2455        public byte[] decrypt(byte[] encryptedData) throws CryptoException {
2456            try {
2457                verifyOperation(OP\_DECRYPT);
2458                return cipher.doFinal(encryptedData);
2459            }
2460            catch (BadPaddingException ex) {
2461                throw new CryptoException("Error with data padding.", ex);
2462            }
2463            catch (IllegalBlockSizeException ex) {
2464                throw new CryptoException("Error with data block size.", ex);
2465            }
2466        }
2467
2468        /**
2469         * Returns the desired encrypted data decrypted using the private key defined
                previously.
2470         *
2471         * @param encryptedData the data to be decrypted.
2472         * @param offset the offset in <tt>encryptedData</tt> where the encrypted data
                starts.
2473         * @param length the encrypted data length.
2474         * @return the decrypted data.
2475         * @throws CryptoException if a padding or block size error occurred.
2476         */
2477        public byte[] decrypt(byte[] encryptedData, int offset, int length) throws
            CryptoException {
2478            try {
2479                verifyOperation(OP\_DECRYPT);
2480                return cipher.doFinal(encryptedData, offset, length);
2481            }
2482            catch (BadPaddingException ex) {
2483                throw new CryptoException("Error with data padding.", ex);
2484            }
2485            catch (IllegalBlockSizeException ex) {
```

```
2486            throw new CryptoException("Error with data block size.", ex);
2487        }
2488    }
2489
2490    /**
2491     * Returns the desired encrypted data decrypted using the private key defined
            previously.
2492     *
2493     * @param encryptedData the data to be decrypted.
2494     * @param offset the offset in <tt>encryptedData</tt> where the encrypted data
            starts.
2495     * @param length the encrypted data length.
2496     * @param output the byte buffer to write the decrypted data.
2497     * @return the amount of decrypted data.
2498     * @throws CryptoException if a padding or block size error occurred.
2499     */
2500    public int decrypt(byte[] encryptedData, int offset, int length, byte[] output)
            throws CryptoException {
2501        try {
2502            verifyOperation(OP\_DECRYPT);
2503            return cipher.doFinal(encryptedData, offset, length, output);
2504        }
2505        catch (IllegalArgumentException ex) {
2506            throw new CryptoException("Error with encrypted data.", ex);
2507        }
2508        catch (BadPaddingException ex) {
2509            throw new CryptoException("Error with data padding.", ex);
2510        }
2511        catch (IllegalBlockSizeException ex) {
2512            throw new CryptoException("Error with data block size.", ex);
2513        }
2514        catch (IllegalStateException ex) {
2515            throw new CryptoException("Wrong state found.", ex);
2516        }
2517        catch (ShortBufferException ex) {
2518            throw new CryptoException("Buffer too small.", ex);
2519        }
2520    }
2521
2522    /**
2523     * Continues a multiple-part encryption or decryption operation (depending on how
            this cipher was initialized),
2524     * processing another data part.
2525     *
```

```
2526         * @param input the input buffer
2527         * @param inputOffset the offset in <tt>input</tt> where the input starts
2528         * @param inputLen the input length
2529         * @param output the buffer for the result
2530         * @param outputOffset the <tt>offset</tt> in output where the result is stored
2531         * @return the number of bytes stored in <tt>output</tt>
2532         * @throws CryptoException if this cipher is in a wrong state or the given output
                 buffer is too small to hold the result.
2533         */
2534        public int addDataToEncrypt(byte[] input, int inputOffset, int inputLen, byte[]
             output, int outputOffset) throws CryptoException {
2535            try {
2536                verifyOperation(OP\_ENCRYPT);
2537                return cipher.update(input, inputOffset, inputLen, output, outputOffset);
2538            }
2539            catch (IllegalStateException ex) {
2540                throw new CryptoException("Wrong state found.", ex);
2541            }
2542            catch (ShortBufferException ex) {
2543                throw new CryptoException("Buffer too small.", ex);
2544            }
2545        }
2546
2547        /**
2548         * @return The asymmetric transformation in use.
2549         */
2550        public String getTransformation() {
2551            return transformation;
2552        }
2553
2554        /**
2555         * @param string The asymmetric transformation to be used.
2556         */
2557        public void setTransformation(String string) throws NoSuchPaddingException,
             NoSuchAlgorithmException {
2558            transformation = string;
2559            cipher = Cipher.getInstance(transformation);
2560            currentOperation = OP\_NONE;
2561        }
2562    }
```

B.1 Como Executar

Como executar a rede de Misturadores JmixNet[2]

O primeiro passo para executar a rede de mistura é estabelecer uma ICP (PKI) simples, na qual cada servidor deve ter um certificado assinado pela Autoridade Certificadora da JMixNet. O nome comum (campo CN) do certificado deve ser o endereço IP do servidor.

Para criar a Autoridade Certificadora da JMixNet você deve:

* Gerar a chave da Autoridade Certificadora da JMixNet (OpenSSL):

openssl req -x509 -newkey rsa:1024 -keyout jmixnet_ca_key.pem -out jmixnet_ca.pem

* Importar o certificado da AC no arquivo "JREHOME/lib/security/cacerts"(Java keytool):

keytool -import -alias jmixnetCA -file jmixnet_ca.cer -keystore cacerts

Para gerar os certificados de servidor (repita estes passos para cada servidor) você deve:

* Gerar o par de chaves do servidor (Java keytool):

keytool -genkey -alias serverKey -keystore JMixNetServer.keystore

* Gerar o Certificate Signing Request do servidor (Java keytool):

keytool -certreq -alias serverkey -keystore JMixNetServer.keystore -file serverReq.csr

* Assinar o certificado do servidor (OpenSSL):

openssl x509 -in serverReq.csr -out jmixnet_server66.cer -days 360 -req -CA jmixnet_ca.pem -CAkey jmixnet_ca_key.pem -CAcreateserial

* Importar o certificado do servidor (Java keytool):

keytool -import -alias serverKey -file jmixnet_server66.cer -keystore JMixNetServer.keystore -trustcacerts

O próximo passo é definir o arquivo de configuração da JMixNet (jmixnet.cfg). Este arquivo deve conter ao menos uma propriedade, a cadeia (chain) de servidores:

[2]Como criar/executar a rede de misturadores JmixNet está Disponível em: ¡http://jmixnet.sourceforge.net/¿ Acesso em: 03 maio 2011 .

Exemplo: chain=192.10.15.66,192.10.15.67,192.10.15.68,192.10.15.69

Esta configuração indica quem é o primeiro servidor, servidores intermediários, e o último servidor. A JMixNet usa a porta TCP 1981 como padrão. A configuração da cadeis pode conter endereços IP ou nomes de domínio, incluindo ”localhost”.

Para executar a rede você precisa iniciar os servidores em ordem reversa, ou seja, iniciar o primeiro servidor, então o anterior, e assim por diante. O primeiro servidor da cadeia será o último a ser iniciado.