UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

Evandro Araujo de Sousa

Software para Assinatura Digital

Trabalho de Conclusão de Curso submetido ao Departamento de Informática e Estatística da Universidade Federal de Santa Catarina para a obtenção do grau de Bacharelado em Sistemas de Informação.

Prof. Ricardo Felipe Custódio, Dr.
Orientador

Florianópolis, Abril de 2006

Software para Assinatura Digital

Evandro Araujo de Sousa

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção do título de Bacharelado em Sistemas de Informação.

_____

Prof. Olinto José Varela Furtado, Dr.

Coordenador do Curso

Banca Examinadora

_____

Prof. Ricardo Felipe Custódio, Dr.

Orientador

_____

Prof. Júlio da Silva Dias

_____

Marcelo Luiz Brocardo

A todos aqueles que apoiaram e incentivaram o meu estudo ao longo do curso. Em especial aos meus familiares que a todo tempo desejam e torcem pelo meu sucesso. Aos amigos que me ajudaram de algum modo no estudo e na obtenção de conhecimento. E uma dedicatória especial ao meu filho Vitor Specht, pois que seja um incentivo e um caminho para a sua formação e para o seu futuro.

Agradecimentos

Ao professor e orientador Ricardo Felipe Cústodio, mestre que ao longo do período de realização deste trabalho de conclusão de curso, direcionou as minhas atividades de maneira precisa e necessária, com o objetivo de orientar na concretização deste projeto.

Ao professor Júlio da Silva Dias pelas revisões e dicas em relação aos tópicos importantes e essencias no desenvolvimento do trabalho.

A Marcelo Luiz Brocardo por aceitar o convite de participar da banca examinadora e pelas dicas do padrão de assinatura digital.

Aos demais mestres, que durante a minha graduação trabalharam em preparar as aulas, apostilas e todo material necessário para transmitir os seus ensinamentos.

Não poderia de deixar os agradecimentos aos meus colegas de turma, fazendo uma justa homenagem à turma SIN 002, na qual tive a oportunidade de aprender com os erros e acertos, não só na vida acadêmica mas também na convivência social.

Aos demais professores e servidores que fazem desta Universidade um excelente centro de ensino para a educação do cidadão brasileiro.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

AC	Autoridade Certificadora
AES	Advanced Encrytion Standard
API	Applications Programming Interface
CRL	Certificate Revocation List
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard
ECDSA	Elliptic Curve Digital Signature Algorithm
GPL	General Public License
HTTPS	HiperText Transfer Protocol Secure
ICP	Infra–estrutura de Chaves Públicas
ICP-Brasil	Infra–estrutura de Chaves Públicas Brasileira
IP	Internet Protocol
ITI	Instituto Nacional de Tecnologia da Informação
JCA	Java Cryptography Architecture
JCE	Java Cryptography Extension
JDK	Java Development Kit
LCR	Listas de Certificados Revogados
LDAP	Lightweight Directory Access Protocol
MD2	Message Digest versão 2
MD5	Message Digest versão 5
MIT	Massachusetts Institute of Technology
NIST	National Institute of Standards and Technology
NSA	National Security Agency
PKCS	Public Key Cryptography Standards
RFC	Request for Comments
RSA	Rivest–Shamir–Adleman
SHA-1	Secure Hash Algorithm
SNTP	Simple Network Time Protocol
URL	Universal Resource Locator

Resumo

Este projeto apresenta uma proposta de software para assinatura digital, fazendo uso da linguagem de programação Java.

Para assinar digitalmente um arquivo é necessário utilizar algoritmos da criptografia assimétrica, a qual utiliza um par de chaves distintas. Diferente do processo de criptografia simétrica, que utiliza apenas uma chave secreta.

A assinatura digital de um arquivo eletrônico, consiste em realizar a função resumo deste arquivo e cifrar o resultado obtido com a chave privada do assinante. Para verificar esta assinatura, é preciso decifrar o resultado com a chave pública, comparando com a função resumo do arquivo original, verificando se são idênticos.

O Java possui uma arquitetura de criptografia, tendo classes e interfaces para a implementação e a utilização de aplicativos na área de segurança digital. Existem alternativas ao provedor de serviços de criptografia da Sun, e um deles é o provedor da BouncyCastle, que oferece o formato PKCS#7 para a assinatura digital.

Existem diversos softwares de assinatura digital, alguns comerciais, outros software livre. A maioria com funcionalidades idênticas, mas que deixam a interoperabilidade em segundo plano.

O software proposto para a assinatura digital possui funcionalidades básicas para assinar, verificar e gerenciar certificados e chaves públicas e privadas.

Palavras Chave: Assinatura Digital, Certificado Digital, Criptografia, Linguagem de Programação Java.

Abstract

This project presents a proposal of software for digital signature, making use of the programming language Java.

To sign digitally an archive is necessary to use algorithms of the asymmetrical cryptography, which uses a pair of distinct keys. Different of the process of symmetrical cryptography, that uses only one secret key.

Digital signature of an electronic archive, consists of carrying through the hash function of this archive and the gotten result to cryptography with a private key of the subscriber. To verify this signature, decryptography the result of the hash function with the public key, comparing hash function with the original archive and verifies if results are identical.

The programming language Java possess a cryptography architecture, for the implementation e use of software in the security area. They exist alternative to the supplier of services of criptografia of the Sun, and the one of them and supplier of the BouncyCastle, that offers format PKCS#7 for the digital signature.

They exist diverse softwares of digital signature, some commercial ones, other free software. The majority with identical functionalities, but that they leave the not operate in second plain.

Considered software for digital signature possess the basic functionalities to sign, to verify and to manage public and private keys and certificates.

Keywords: Digital Signature, Digital Certificate, Cryptography, Programming Language Java.

Capítulo 1

Introdução

Com o crescente uso do meio eletrônico, podendo ser citado a internet, uma rede de computadores e o uso de correspondências eletrônicas, torna-se cada vez mais fácil a substituição do uso de documentos em papel pelos documentos eletrônicos, com isso é necessário garantir a segurança da tramitação destes documentos, ou seja, precisa garantir que aquilo que o remetente redige, será o mesmo que o destinatário lê. Esta garantia pode ser obtida através da assinatura digital, tornando o documento eletrônico seguro, íntegro e autêntico para o receptor.

Conceitua-se assinatura digital como sendo um mecanismo digital utilizado para fornecer confiabilidade, sobre a autenticidade de um documento eletrônico e sobre o remetente do mesmo [VOL 01]. No entanto, para que os documentos eletrônicos sejam técnica e juridicamente seguros é necessário que eles sejam assinados e datados digitalmente, o que garante os requisitos de segurança [COS 03].

A assinatura consiste na expressão da vontade ou do consentimento do assinante em relação ao conteúdo do documento. Deve haver, portanto, uma conexão entre o conteúdo e o assinante [CUS 03]. A funcionalidade da assinatura digital ocorre atráves dos algoritmos de autenticação, ou seja, efetua-se um processo lógico-matemático sobre a mensagem, obtendo-se uma determinada expressão que será utilizada como assinatura digital [VOL 01].

A partir da existência da legislação que tornou válido os documentos

eletrônicos, o interesse pelos mesmos aumentaram. O documento assinado eletronicamente é reconhecido da mesma forma que um documento assinado no papel, conforme a Medida Provisória nº 2.200, de 02 de agosto de 2001 [dRCCSpAJ 05], a qual regula o sistema nacional de certificação digital, com o objetivo de validar juridicamente documentos eletrônicos. No uso de assinaturas digitais é verificado a existência de dois modelos de leis, que tratam a questão da inversão do ônus da prova, os quais são citados abaixo:

- modelo de lei da Uncitral: o assinante deve provar que a assinatura não foi de sua autoria ou foi obtida de forma fraudulenta. A legislação brasileira tem semelhanças com este modelo;
- diretiva da comunidade européia: a parte interessada deve apresentar evidências que o assinante realizou a assinatura.

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo geral deste trabalho de conclusão de curso é a apresentação de uma proposta de software para assinatura digital.

1.1.2 Objetivos Específicos

Este projeto tem como objetivos específicos:

- Apresentar o contexto da assinatura digital;
- Analisar as formas legais de utilizar a assinatura digital em documentos eletrônicos, conforme prescreve a infra-estrutura de chaves públicas brasileira;
- Estudar as técnicas de criptografia utilizadas na assinatura digital;
- Estudar as classes e os pacotes da linguagem de programação Java, que podem auxiliar na implementação da assinatura digital;

- Pesquisar, analisar e apresentar as características de alguns softwares de assinatura digital;

- Apresentar uma proposta de um software para assinatura digital;

- Desenvolver um protótipo da proposta apresentada, implementando-o através da linguagem de programação Java.

1.2 Motivação

O acesso aos documentos eletrônicos de maneira rápida, através das redes de computadores, em especial a internet, tornou fácil a troca de informações entre pessoas, empresas e instituições, que cada vez mais, substituem a correspondência em papel pela eletrônica.

A motivação deste projeto é apresentar uma proposta de ferramenta desenvolvida na linguagem de programação Java, que seja prática e de fácil uso, para os usuários que utilizam o meio eletrônico com o objetivo de envio de documentos, sejam eles administrativos, jurídicos e/ou pessoais, tornando seguro o seu envio e recebimento, e também produzindo seus efeitos legais, do mesmo modo que a documentação em papel.

O fato de se escolher a linguagem de programaçõ Java para o desenvolvimento do software foi motivado devido a esta linguagem oferecer classes e pacotes para a implementação da criptografia, assinatura digital e o gerenciamento de chaves criptográficas e certificados digitais.

1.3 Conteúdo do Projeto

A partir da introdução relatada, este projeto é constituído do capítulo 2 onde são estudadas e apresentadas as diversas técnicas de criptografia, utéis para o entendimento da funcionalidade da assinatura digital, além de sua utilização na geração de chaves criptográficas.

No capítulo 3 é contextualizado a assinatura digital, descrevendo suas características e vantagens, além de mencionar o teor da legislação que trata do reconhecimento dos documentos eletrônicos para os devidos fins legais. Também trata da infra-estrutura das chaves públicas brasileiras e dos certificados digitais, com suas principais informações, devidos cuidados na sua utilização e ainda a forma de obtenção junto à uma Autoridade de Registro.

Já no decorrer do capítulo 4 são vistos os fundamentos da linguagem Java, suas classes e pacotes para implementações de criptografia, certificados e assinaturas digitais.

O capítulo 5 é destinado a pesquisa e apresentação de softwares de assinatura digital, com a intenção de relatar o funcionamento, preço de aquisição e licensa de uso, vantagens e desvantagens, enfim realizar um estudo comparativo com o objetivo de obter dados para se propor um novo software de assinatura digital.

É descrito no capítulo 6 a proposta de construção para um software de assinatura digital, com o escopo e os requisitos para atender o que se propõe. E consequentemente no capítulo 7, são demonstradas as funcionalidades do software e implementações do código-fonte.

Finalmente o capťulo 8 é destinado as conclusões do projeto desenvolvido, apresentando as considerações finais e sugestões para trabalhos futuros.

Capítulo 2

Criptografia

2.1 Introdução

Este capítulo tem como objetivo o estudo da criptografia, em especial a forma de utilização dos algoritmos criptográficos para a implementação da assinatura digital. Através deste estudo será possível compreender como a criptografia é utilizada para a geração da assinatura digital em um documento eletrônico.

É objetivo deste estudo os conceitos básicos da criptografia, os quais atualmente estão sendo amplamente utilizados para a assinatura digital. Não faz parte do objetivo deste capítulo o estudo detalhado dos algoritmos criptográficos e as suas funções matemáticas.

A criptografia é a ciência de segurança de informações [THI 03]. E utiliza qualquer um dos vários métodos para transformar arquivos legíveis em algo ilegível [BUR 02], incluindo tecnologias, como mescla e/ou substituições de palavras, com o propósito de ocultar informações.

O texto legível é convertido em um equivalente codificado, chamado texto cifrado, por um algoritmo de codificação, depois o texto cifrado é decodificado pelo receptor usando uma chave de decodificação [FRE 95]. Ou seja, um algoritmo de criptografia serve para cifrar os dados e ao ser inserido uma chave criptográfica, o algoritmo realiza seus passos utilizando esta chave para alterar o texto legível e convertê-lo em um

texto cifrado. Para decifrar o texto utiliza-se uma chave e assim o algoritmo inverte os passos e converte o texto cifrado de volta no texto legível original [BUR 02].

Um dos critérios de projeto de um sistema criptográfico é que este deve permanecer seguro mesmo quando os algoritmos de ciframento e deciframento sejam conhecidos [LUC 86], por esta razão é que são utilizados as chaves criptográficas.

Mensagem segura é uma mensagem que possui resistência aos possíveis ataques, e a criptografia é a arte e a ciência de manter esta mensagem com esta característica, apresentando as seguintes funções importantes:

- confidencialidade - garantia de acesso à mensagem apenas por elementos autorizados;
- autenticidade - certifica a origem de uma mensagem;
- integridade - possibilita ao receptor verificar se a mensagem recebida não sofreu alterações;
- não-repúdio - emissor não pode negar uma mensagem por ele enviada;
- tempestividade - prova a existência de uma mensagem em determinada data e hora sem sofrer alteração desde aquele momento.

Na obtenção da mensagem segura utiliza-se processos de cifrar e decifrar com o uso de algoritmos criptográficos, os quais são baseados em substituições, funções e operadores matemáticos, transformando textos originais em textos cifrados.

A idéia fundamental da criptografia é tornar computacionalmente impraticável, para uma pessoa não autorizada, recompor o texto original em posse do texto cifrado [RIB 03].

O algoritmo criptográfico tem sua segurança nas chaves criptográficas, sendo simétricas, onde a mesma chave é utilizada tanto para cifrar como para decifrar ou sendo assimétricas, que utilizam chaves diferentes para cifrar e decifrar os documentos eletrônicos.

A criptografia é uma ferramenta necessária para assegurar segurança dos dados, tornando capaz o armanezamento e a transferência de informações confidenciais, sendo a prova de fator crítico no sucesso de transações eletrônicas. Contudo a criptografia não resolverá todos os problemas de segurança e além do mais não deixará de ser tolerante às falhas.

Nas próximas seções é descrito os dois processos de criptografia, sendo que na seção 2.2 a criptografia simétrica e na seção 2.3 a criptografia assimétrica. A seção 2.4 é destinada aos algoritmos de função resumo. O capítulo é finalizado com a seção 2.5 que trata de algoritmos para assinatura digital.

2.2 Criptografia Simétrica

Utilizada nos algoritmos criptográficos que fazem uso da mesma chave nos processos de cifrar e decifrar um texto. A chave deve ser secreta e de exclusivo conhecimento do emissor e receptor das mensagens.

No algoritmo de chave simétrica é utilizado funções matemáticas bem complexas e com características bem específicas, a exemplo pode-se citar uma função unidirecional, que é uma função que passa a ser satisfeita se for computacionalmente viável computá-la e computacionalmente inviável computar a sua inversa [LUC 86]. Estas funções tem como objetivo dificultar ao máximo as tentativas de recuperação do texto original por pessoas não autorizadas. A diferença entre as diversas funções matemáticas usadas para cifrar e decifrar mensagens é o que define e caracteriza um algoritmo criptográfico [RIB 03].

Para este algoritmo é necessário ter uma comunicação segura, porque o segredo da chave depende do envio da mesma para o receptor, pois o emissor ao cifrar seu texto com uma chave, é obrigado enviá-la para que seja utilizada para decifrar o texto cifrado recebido pelo receptor. Caso a comunicação não seja segura e no envio da chave simétrica um intruso vier a obté-la, este terá acesso ao texto legível utilizando a chave capturada ao decifrar o texto cifrado.

O principal problema relacionado à criptografia simétrica está no fato

de que o emissor e os receptores devem ter acesso à mesma chave [PAS 02]. Portanto é necessário adotar uma segurança para a troca e a guarda da chave, processo conhecido como gerenciamento de chave.

Nas subseções seguintes serão descritos características e peculiaridades de alguns algoritmos de chave simétrica, tais algoritmos são de conhecimento amplo e muito utilizados no ciframento e deciframento de documentos eletrônicos.

2.2.1 DES

O padrão de criptografia de dados (DES - Data Encryption Standard) é um método de criptografia amplamente utilizado [THI 03]. Originou-se na IBM em 1977 e foi adotado pelo Departamento de Defesa dos Estados Unidos.

O DES é uma cifragem de blocos[1] que utiliza uma chave de 56 bits realizando manipulações de bits sobre o texto legível e, para decifrar o texto cifrado, simplesmente reverte tudo.

Com o avanço tecnológico tornando os computadores cada vez mais velozes no seu desempenho e consequentemente nos processamentos dos dados, o DES acabou se tornando um algoritmo possível de quebra pela força bruta, deixando sua eficiência a desejar.

Em 1999, na RSA Conference, a Eletronic Frontier Foundation quebrou uma chave de DES em menos de 24 horas [BUR 02]. Com isto depois de um tempo surgiu um novo padrão, o AES - padrão avançado de criptografia.

2.2.2 Triple DES

O Triple DES é um substituto ao DES, também sendo amplamente utilizado. Ele realiza três vezes o algoritmo de DES.

Entretanto este algoritmo apresenta dois problemas. Um deles é que os

[1] Uma cifragem de blocos opera sobre blocos de dados. Quando é enviado ao algoritmo um fragmento de dados para cifrar ou decifrar, ele divide o texto simples em blocos e opera sobre cada bloco de maneira independente [BUR 02].

analistas criptográficos descobriram uma maneira de simplificar o ataque de força bruta, pois é errado pensar na necessidade de um ataque de 168 bits, porque existe maneiras inteligentes para reduzir isso a um ataque de força bruta de 108 bits. O outro problema esta relacionado à velocidade, se o DES leva muito tempo para cifrar ou decifrar dados, então o Triple DES acaba sendo três vezes mais lento que o algoritmo DES [BUR 02].

2.2.3 AES

O padrão de criptografia avançada (AES - Advanced Encryption Standard) é esperado que se torne um padrão mundial [BUR 02].

A sua origem foi devido a fragilidade do algoritmo DES, que fez com que o governo americano lançasse um concurso para definição de um novo padrão de criptografia simétrica. O algoritmo vencedor foi o Rijndael, assim definindo o novo padrão de criptografia simétrica, sendo chamado de AES [PAS 02].

O AES suporta tamanho de chave e de bloco de 128 a 256 bits. Caso a Lei de Moore (a cada 18 meses a tecnologia dobra o poder computacional dos semicondutores) não seja contrariada e não haja nenhuma descoberta científica relevante (matemática, física, etc.) então o AES/128 bits durará até 2066 e o AES/256 alguns séculos [BUA 03].

2.3 Criptografia Assimétrica

A criptografia assimétrica é um sistema criptográfico que envolve o uso de um par de chaves matematicamente relacionadas, uma chave pública e outra privada [dSD 04]. Também chamada de criptografia de chave pública. Usa duas chaves distintas, uma é disponibilizada para acesso de qualquer um (chave pública) e a outra mantida em segredo (chave privada). O texto cifrado com a chave privada somente é decifrado com a chave pública e vice-versa, sendo possível a ocorrência do contrário, ou seja, o texto legível pode ser cifrado pela chave pública e o resultado, será decifrado com a chave privada.

Quando a chave de deciframento é distinta da chave de ciframento,

e ainda mais, é uma função computacionalmente intratável (sem solução matemática) daquela chave, dizemos que o sistema criptográfico é assimétrico [LUC 86].

Os algoritmos de criptografia assimétrica surgiram com o objetivo de solucionar alguns problemas que envolvem a criptografia simétrica. Primeiramente o gerenciamento de chaves, em especial a sua distribuição, pois na criptografia simétrica existe a necessidade de um canal seguro para distribuir a chave utilizada no deciframento de um texto cifrado. Diferente na criptografia assimétrica, onde existe a chave pública de conhecimento de todos, a qual pode ser usada para cifrar ou decifrar um texto. E em termos de funções da criptografia, possui autenticidade e confidencialidade, características não existentes na criptografia simétrica. A autenticidade é garantida quando o texto é cifrado com a chave privada do emissor, assim o receptor pode verificar a identificação do emissor no deciframento do texto cifrado. Já a confidencialidade é obtida quando o texto é cifrado com uma chave pública do receptor, pois somente ele poderá decifrar este texto.

Ao comparar o uso dos algoritmos simétricos com o uso dos algoritmos assimétricos, observa-se que o algoritmo simétrico obtém um processamento mais rápido, já o algoritmo assimétrico permite o envio da chave através do mesmo canal de comunicação da mensagem, facilitando o gerenciamento das chaves [VOL 01].

O algoritmo comumente utilizado na criptografia assimétrica é o RSA, desenvolvido por Ron Rivest, Adi Shamir e Len Adleman, que publicaram-no em 1978. Neste algoritmo o problema intratável é a fatoração de inteiros [LUC 86]. O emissor computa o produto de dois primos inteiros e publica este produto, o resultado é a chave pública. Para decifrar textos cifrados com a chave pública, é preciso a fatoração daquele produto de dois primos, esta fatoração e a chave privada estão de posse do emissor.

O RSA não é rápido quanto os algoritmos simétricos, o RC4 (provavelmente o algoritmo simétrico mais rápido amplamente utilizado hoje em dia) cifrará os dados a uma taxa de 700 vezes mais rápida do que os 1024 bits do RSA (lembrando que 1024 bits é o tamanho de chave mais comum utilizado no RSA) [BUR 02].

2.4 Função Resumo

Esta função conhecida também como função hash, tem a funcionalidade de gerar um resumo de um arquivo qualquer. Com o resultado gerado é possível garantir a integridade de uma mensagem [MIG 02], porque o resumo obtido no destino deve ser igual ao resumo obtido na origem. Caso haja alguma alteração durante o envio da mensagem, o resultado obtido no destino será diferente daquele que se teve na origem da mensagem.

A função resumo tem duas propriedades importantes [KAZ 03], a impossibilidade de fazer a operação inversa, ou seja, dado um resumo deve ser computacionalmente inviável obter a mensagem original. E a diferenciação na geração do resumo, pois duas mensagens distintas não devem originar um resumo idêntico. Além destas citadas é relevante dizer que com um bom algoritmo de função hash não se pode localizar nenhuma mensagem que produza um resumo em particular.

Dois dos mais conhecidos algoritmos hash são [VOL 01]:

- MD5, desenvolvido pelo professor Ron Rivest, do MIT. Aceita mensagens de qualquer tamanho e produz um bloco de 128 bits.
- SHA - Secure Hash Algorithm, desenvolvido pelo NIST e pelo NSA. Aceita mensagens de comprimento inferior a 2^{64} e produz um resumo de 160 bits.

2.5 Algoritmos para Assinatura Digital

A assinatura digital é a criptografia resultante do processo de cifragem de um determinado bloco de dados (documento) pela utilização da chave privada de quem assina em um algoritmo assimétrico. A verificação desta assinatura é feita decifrando a criptografia resultante com a chave pública correspondente de quem assinou o bloco de dados [MOR 05].

A autenticidade, convence o receptor que o documento foi assinado pelo emissor, a integridade garante que após o documento ter sido assinado não pode ser alterado e não tem como repudiar o envio do documento, negando que não foi o emissor

quem assinou o documento. Estas são características da assinatura digital. Sendo que a assinatura não pode ser removida do documento e passada para outro, pois só é válida para o documento que inicialmente foi utilizado.

Para garantir estes benefícios da criptografia, a assinatura digital se utiliza de alguns algoritmos criptográficos, sendo que sua implementação é feita através do uso de funções resumos (algoritmos MD5 ou SHA1) ou utilizando o DSS, tudo embasado na criptografia assimétrica. A seguir é descrito os dois mais importantes e conhecidos algoritmos utilizados na assinatura digital.

- RSA - algoritmo utilizado para cifrar um resumo (resultado da função hash aplicada no documento eletrônico original) com uma chave privada. A verificação é realizada com o deciframento do resumo criptografado utilizando-se a chave pública correspondente à chave privada, o resultado é comparado com a aplicação da função resumo no documento eletrônico original.

- DSA - o documento eletrônico original é resumido com o algoritmo SHA1, tratando o resultado como um número de 160 bits de comprimento. Outro número, aleatório ou pseudo-aleatório, chamado de **k**, é enviado ao algoritmo. Finalmente com os números anteriores e a chave privada, o algoritmo realiza operações matemáticas que resultam na assinatura digital, ou seja, em dois números chamados **r** e **s**. O processo de verificação é realizado através de operações matemáticas com o resumo do documento, a chave pública e o número **s** que resultam no novo número **v** que deve ser idêntico ao número **r** [BUR 02].

Capítulo 3

A Assinatura Digital

3.1 Introdução

Os mecanismos de assinatura digital foram criados com o objetivo de substituir a assinatura manuscrita, por uma que levasse para o mundo digital as mesmas garantias do mundo real. A simples digitalização da imagem da assinatura manuscrita não é suficiente para alcançar esse propósito, pois a mesma pode ser copiada e anexada em qualquer documento tornando-a fácil de ser forjada.

A assinatura manuscrita, um sinal gráfico pessoal, é rabiscado em um documento para ratificar o seu conteúdo. É válida porque está atrelada ao papel de forma permanente. A mesma não pode ser transferida para uma outra folha. Mas sendo possível de falsificação pois o sinal gráfico pode ser copiado de forma idêntica por uma pessoa habilidosa.

É necessário distinguir assinatura digital de assinatura digitalizada. A assinatura digitalizada é a reprodução do sinal gráfico como imagem por um equipamento do tipo escaneador. Ela não garante a autoria e integridade do arquivo eletrônico, porque não existe uma associação inequívoca entre o assinante e o texto digitalizado, uma vez que pode ser facilmente copiada e inserida em outro arquivo [dTdI 05].

3.2 Características

A assinatura digital é uma assinatura eletrônica que pode ser usada para autenticar a identidade do remetente de uma mensagem, assegurando que o conteúdo original da mensagem não será alterado [THI 03].

O processo de assinatura digital utiliza algoritmos de criptografia assimétrica, obtida através do ciframento do arquivo a ser assinado com a chave privada do assinante. O processo para gerar assinatura digital é muito simples e pode ser aplicado não somente a um texto, mas também a qualquer tipo de arquivo digital (imagem, som, e-mail, etc).

A assinatura digital fica de tal modo vinculada ao arquivo eletrônico que, ante a menor alteração neste, a assinatura se torna inválida, por exemplo, a inserção de mais um espaço entre duas palavras [dTdI 05]. Assim é possível verificar a integridade do documento, permitindo ao destinatário perceber qualquer modificação feita no documento original.

Como o resultado de cifrar um arquivo é o mesmo tamanho do arquivo original e o tempo para cifrar depende do tamanho do arquivo, seria inviável apenas cifrar o arquivo original para se obter a assinatura digital. Assim o que se faz é aplicar uma função resumo no arquivo e posteriormente cifrar o resumo e não o arquivo original. Com isso a verificação da assinatura se dá na comparação entre o resumo do arquivo original e do arquivo recebido.

A assinatura digital não pode ser extraída de um documento e inserida em outro, pois só é válida para o documento que inicialmente foi utilizado [CUS 03]. Portanto uma característica da assinatura digital é ser única para apenas um documento, mesmo que o remetente seja o próprio para diversos documentos.

Um documento ao ser assinado digitalmente produz consigo a prova de autoria. A comprovação de autoria é outra característica da assinatura digital, que convence o destinatário de que o documento recebido foi realmente assinado pelo emissor, pois após assinado não pode ser alterado.

O emissor não tem como repudiar o envio do documento, ou seja, dizer

que não foi ele quem assinou o documento, pois desde que não aconteça algo de anormal, o emitente é a única pessoa que tem acesso à chave privada que gerou a assinatura [dTdI 05].

3.3 Funcionamento

O processo de assinatura digital de um documento eletrônico é realizado da seguinte maneira:

- realizar a função resumo no documento que deseja assinar;
- cifrar o resumo com a chave privada do remetente;
- enviar o documento eletrônico, junto com a assinatura digital.

De modo inverso, para que se realize o processo de verificação de uma assinatura digital em um documento eletrônico recebido, deve-se seguir as etapas:

- decifrar a assinatura digital com a chave pública do remetente;
- realizar a função resumo no documento original recebido;
- comparar o resultado do deciframento da assinatura digital com o resultado da função resumo;
- a assinatura digital é válida se a comparação dos resultados forem idênticos.

3.4 Vantagens

As vantagens obtidas ao utilizar uma assinatura digital na produção de documentos eletrônicos são:

- envio por uma rede de computadores, inclusive a internet, e verificado remotamente através de uma cópia do mesmo documento.

- na assinatura manuscrita é comparado apenas o sinal gráfico, não sendo importante o conteúdo do papel, já na assinatura digital, o seu reconhecimento sempre envolve o conteúdo do arquivo, pois qualquer adulteração do arquivo original torna a verificação da assinatura digital incorreta.

- conferir maior grau de segurança, garantindo ao destinatário integridade e autenticidade do documento, ao contrário dos documentos não assinados que possuem características de alterabilidade e de fácil falsificação.

3.5 Aplicabilidade

A utilização da assinatura digital é possível em diversas aplicações que exigem o uso de autoria e integridade de dados. Com o constante aumento de serviços disponíveis pela rede mundial de computadores, exige-se cada vez mais a aplicação de assinatura digital na disponibilidade de tais serviços.

Entre as diversas aplicações possíveis, encontram-se as seguintes [dTdI 05]:

- comércio eletrônico;
- processos judiciais e administrativos em meio eletrônico;
- assinatura da entrega da declaração de renda;
- obtenção e envio de documentos cartorários;
- transações seguras entre instituições financeiras;
- Diário Oficial eletrônico;
- correio eletrônico.

3.6 Legalidade

A distância entre o ritmo da evolução tecnológica e o ritmo da evolução legal para atender à tecnologia é excessivamente grande [VOL 01]. Isto acaba resultando

em situações complexas, onde é possível a aparição de um eventual crime sem a existência de uma legislação que enquadre determinada atividade ilícita.

Para o caso da assinatura digital esta disparidade apresenta pontos ainda mais agravantes, onde a necessidade de estar devidamente regulamentada torna-se o ponto de partida para a sua completa utilização, pois sem sua validade jurídica, o uso da assinatura digital fica restrita a poucos serviços.

O documento assinado digitalmente é reconhecido da mesma forma que um documento assinado de próprio punho, conforme consta no § 1º, do Art 10 da Medida Provisória nº 2.200-2, de 24 de agosto de 2001, transcrito a seguir: "As declarações constantes dos documentos em forma eletrônica produzidos com a utilização de processo de certificação disponibilizado pela ICP-Brasil presumem-se verdadeiros em relação aos signatários, na forma do Art 131 da Lei nº 3.071, de 01 de janeiro de 1916 - Código Civil."[dRCCSpAJ 05].

Pode-se utilizar outro meio de comprovação da autoria e integridade de documentos eletrônicos por certificados não emitidos pela ICP-Brasil, desde que válido pelas partes envolvidas.

3.7 Infra-Estrutura de Chaves Públicas Brasileira

Para garantir a autenticidade, a integridade e a validade jurídica dos documentos eletrônicos, das aplicações de suporte e das aplicações que utilizam certificados digitais, assim como a realização de transações eletrônicas seguras, é instituído a Infra-Estrutura de Chaves Públicas Brasileira, a ICP-Brasil.

A ICP-Brasil é composta por uma cadeia de autoridades certificadoras, formada por uma Autoridade Certificadora Raiz (AC-Raiz), sendo o Instituto Nacional de Tecnologia da Informação - ITI, formada também por Autoridades Certificadoras (AC) e Autoridades de Registro (AR) e o Comitê Gestor, que é a autoridade gestora de políticas.

O Comitê Gestor é composto por representantes da sociedade civil e de alguns órgãos públicos, com competência para determinar as políticas a serem executadas pela AC-Raiz.

As Autoridades Certificadoras são credenciadas à AC-Raiz, competindo-lhes o gerenciamento de certificados digitais, através da emissão destes certificados vinculando pares de chaves criptográficas ao titular. Também é de sua competência publicar as listas de certificados revogados, colocando-as à disposição dos usuários.

As Autoridades de Registro também são credenciadas às AC-Raiz e estão vinculadas operacionalmente a uma determinada AC. Tem como competência a identificação e o cadastro de usuários, os quais deverão estar presentes para realizarem a solicitação do certificado. Feita a solicitação, cabe à AR encaminhá-las às AC.

3.8 Certificado Digital

O certificado digital é um arquivo que contém informações a respeito do dono do certificado e da AC que emitiu o mesmo, além de outros dados relevantes para o seu uso. Tem como finalidade certificar que a chave pública pertence realmente a uma determinada pessoa.

Para a aquisição de um certificado digital, é necessário que o interessado se dirija a uma Autoridade de Registro, para que se realize a sua identificação mediante a apresentação de diversos documentos pessoais.

Principais informações que constam num certificado digital [dTdI 05]:

- chave pública do emissor;
- nome e endereço de e-mail;
- período de validade do certificado;
- nome da AC que emitiu o certificado;
- número de série do certificado digital;
- assinatura digital da AC.

Figura 3.1: Visualizador de Certificados

Conforme exibido na figura 3.1, pode-se verificar diversas informações do certificado. Inicialmente a devida finalidade que o certificado foi emitido, o seu uso, seguido das informações do dono e do emissor do certificado, tendo ainda a sua validade e por fim as assinaturas pela AC.

Devidos cuidados devem ser tomados com o uso e o armazenamento do certificado digital. Não se deve compartilhar a senha de acesso da chave privada, quanto menos compartilhar a própria chave privada. O armazenamento da chave privada deve ser realizada num dispositivo removível, tipo smart card ou token, assim evitando o seu armazenamento em disco rígido.

A vantagem de se utilizar um certificado emitido no âmbito da ICP-

Brasil é a sua eficácia jurídica, pois ao assinar digitalmente um documento eletrônico, este terá a mesma validade jurídica dos documentos escritos com assinaturas manuscritas. Mas no caso de um certificado estar fora do âmbito da ICP-Brasil a validade jurídica dependerá da aceitação das partes envolvidas.

Capítulo 4

Linguagem de Programação Java

4.1 Introdução

Os projetistas da linguagem de programação Java iniciaram com o C++, removeram, mudaram e adicionaram funcionalidades. A linguagem resultante oferece grande parte do poder e flexibilidade do C++, mas em uma linguagem menor, mais simples e mais segura [SEB 03]. A adição da concorrência aumenta o escopo das aplicações que podem ser escritas em Java, o mesmo acontecendo com as bibliotecas de classes para *applets*, interfaces gráficas, acesso a banco de dados e a rede de computadores.

A linguagem de programação Java foi anunciada em 1995 pela Sun Microsystems, a partir dai criando grande interesse por causa da programação para a internet. Atualmente a linguagem esta sendo utilizada para a criação de páginas com conteúdo interativo e dinâmico, desenvolver aplicativos corporativos e fornecer aplicativos para dispositivos de uso do consumidor final, tais como telefones celulares, eletrodomésticos e eletrônicos.

O uso do Java cresceu mais rapidamente do que o de qualquer outra linguagem de programação. Uma das razões para isso é seu valor para programar páginas da internet. A outra é que o sistema compilador/interpretador para o Java tem sido gratuito e de fácil obtenção [SEB 03], podendo ser adquirido no endereço http://java.sun.com/. Para escrever, compilar e rodar programas é necessário utilizar o Kit de Desenvolvimento

Java, um ambiente que possui todas as ferramentas necessárias e todos os elementos da plataforma Java.

A linguagem é completamente orientada a objetos com forte suporte para técnicas adequadas de engenharia de software [DEI 01]. Apresenta uma biblioteca de classes extensa, que auxiliam o programador nas diversas tarefas de implementação de programas. Portátil, pois é independente de plataforma, podendo ser executado em diversas arquiteturas de hardware e com qualquer sistema operacional sem a necessidade de recompilação, sendo apenas necessário a máquina virtual Java.

Esta introdução teve o objetivo de descrever brevemente os recursos de Java e as vantagens que levaram a escolher esta linguagem de programação para o projeto do software para assinatura digital. Nas próximas seções serão abordados as diversas classes que a linguagem disponibiliza na sua API de segurança, para o desenvolvimento e a utilização de criptografia, assinatura digital e o gerenciamento de chaves públicas e privadas e de certificados digitais.

4.2 Arquitetura de Criptografia Java

A API de segurança faz parte do núcleo de API's da linguagem de programação Java, sendo construida em torno do pacote java.security e de seus subpacotes, fornecendo as classes e as interfaces para a estrutura de segurança em Java. Esta API é projetada para permitir a funcionalidade de baixo e alto nível de segurança no desenvolvimento de programas.

A arquitetura que forma a base da API de segurança em Java é baseada nas classes Provider e Security, que juntas formam um conjunto de mapeamentos que permite à API de segurança determinar dinamicamente o conjunto de classes que deve usar para implementar certas operações [OAK 99].

A classe Security centraliza todas as propriedades e métodos comuns de segurança. E uma de suas finalidades é o controle dos provedores de serviços de criptografia. Abaixo é relacionado a tabela 4.1 com os principais métodos desta classe.

- static int addProvider(Provider provedor) adiciona um provedor no fim da lista dos provedores instalados. É retornado a posição em que o provedor foi instalado ou -1 se não instalado
- static int insertProviderAt(Provider provedor, int posicao) adiciona um provedor na posição especificada. Retorna o mesmo do método anterior
- static Provider[] getProviders() retorna um vetor contendo todos os provedores instalados
- static Provider getProvider(String nomeProvedor) retorna o provedor encontrado pelo parâmetro nomeProvedor
- static void removeProvider(String nome) remove o provedor de nome especificado

Tabela 4.1: Principais métodos da classe Security

Já a classe Provider representa um provedor para a API de segurança Java, ou seja, um provedor de serviços de criptografia que fornece algoritmos, geração de chaves e demais funcionalidades, permitindo implementações específicas.

Os serviços que um provedor pode executar incluem algoritmos, tais como DSA, RSA, MD5 ou SHA-1, geração, conversão e facilidades de gerenciamento de chaves criptográficas. Cada provedor tem um nome e um número de versão e pode ser configurado em tempo de execução de um programa.

O exemplo a seguir, é uma classe que utiliza o pacote java.security e as classes Security e Provider. Ao ser executado, obtém um array de objetos Provider e imprimi o nome e a versão de todos os provedores instalados, através do método toString() da classe Provider.

```
1  import java.security.Security;
2  import java.security.Provider;
3
4  public class Provedores {
5   public static void main(String[] args) {
```

```
6    Provider[] provedores = Security.getProviders();
7    for (int i = 0; i < provedores.length; i++) {
8     System.out.println(provedores[i].toString());
9    }
10   }
11 }
```

Após a execução da classe Provedores obtém-se o seguinte resultado, conforme os provedores instalados no computador que foi executado o programa:

```
SUN version 1.5
SunRsaSign version 1.5
SunJSSE version 1.5
SunJCE version 1.5
SunJGSS version 1.0
SunSASL version 1.5
```

É de se observar que na instalação de um provedor, o mesmo ocupa uma posição num vetor. Quando a classe Security é implementada para utilizar um algoritmo em particular, o vetor de provedores é pesquisado para obter um que possa fornecer o algoritmo solicitado.

4.3 Classe KeyStore

Esta classe representa uma facilidade no armazenamento para chaves criptográficas e certificados. E tem como finalidade o gerenciamento de chaves e de certificados digitais.

Um sistema de gerenciamento de chaves fornece a chave privada do assinante para ele criar uma assinatura digital e fornece a chave pública para a verificação. Além das chaves, este sistema trabalha também com os certificados. Este sistema de gerenciamento é representado por um arquivo e por padronização chamado de keystore. O JDK disponibiliza um utilitário, o keytool, que manipula o arquivo keystore, gerenciando as chaves criptográficas e os certificados. O estudo deste utilitário será apresentado na seção 4.7.

A classe KeyStore controla dois tipos de entradas:

- KeyEntry este tipo de entrada é associada a um par de chaves pública e privada, as quais são armazenadas em um formato protegido. É acompanhada também por um certificado correspondente para a chave pública.

- TrustedCertificateEntry este tipo de entrada contém um certificado com a chave pública de uma entidade.

Existe duas maneiras de instanciar um objeto da classe KeyStore. Uma fornecendo o tipo padrão da keystore, a outra fornecendo um tipo específico. Isto se faz através do método getInstance:

- public static KeyStore getInstance(String nomeKeyStore)

para fornecer o tipo padrão deve-se utilizar o método getDefaultType(), da própria classe KeyStore, no parâmetro do método

para um tipo específico pode-se utilizar “JKS” que é implementado pelo provedor da Sun, “PKCS12” implementado pelo provedor SunJSSE, “BC”implementado pelo provedor da BouncyCastle[1] ou outro qualquer

Após o objeto da classe ser instanciado, é necessário carregá-lo pelo método load, para que se possa utilizá-lo posteriormente.

- public final void load(InputStream fluxoDados, char[] senha)

carrega através de um fluxo de dados que representa o arquivo da keystore e da sua senha

Os métodos abaixo relacionados são utilizados para diversas finalidades, como verificação, obtenção, remoção e inserção de chaves e certificados, além de obter a quantidade total de entradas armazenadas e verificar se um pseudônimo realmente existe no armazenamento:

- public final Enumeration aliases()

retorna uma enumeração dos pseudônimos existentes no armazenamento

- public final boolean containsAlias(String pseudonimo)

[1]Provedor estudado na seção 4.6

verifica se o pseudônimo existe no armazenamento

- public final int size()

retorna o número total de chaves e certificados armazenados

- public final void deleteEntry(String pseudonimo)

remove uma chave ou certificado armazenado que possuia o mesmo pseudônimo do contido no parâmetro

- public final Certificate getCertificate(String pseudonimo)

obtém o certificado armazenado que possui o pseudônimo do parâmetro

- public final void setCertificateEntry(String pseudonimo, Certificate certificado)

armazena um certificado

- public final Key getKey(String pseudonimo, char[] senha)

obtém a chave privada armazenada que possui o pseudônimo do parâmetro

-public final void store(OutputStream fluxoDados, char[] senha)

armazena o fluxo de dados num arquivo, que será um tipo de keystore, juntamente com a senha

4.4 Classe CertificateFactory

Esta classe define as funcionalidades para gerar tipos de certificados, podendo ser um simples certificado, uma cadeia de certificados ou um objeto que representa uma lista de certificados revogados.

Da mesma forma que acontece com objetos da classe KeyStore, primeiro é preciso instanciar um objeto da classe CertificateFactory com o método getInstance, para posteriormente gerar um tipo de certificado que se deseja. Abaixo é demonstrado a sintaxe do método:

- public static final CertificateFactory getInstance(String tipo)

o tipo no parâmetro do método deve ser “X.509”

Com um objeto CertificateFactory instanciado, é possível gerar certificados conforme os métodos abaixo relacionados:

- public final Certificate generateCertificate(InputStream fluxoDados)

carrega através de um fluxo de dados, o qual representa um arquivo de certificado

- public final CRL generateCRL(InputStream fluxoDados)

carrega através de um fluxo de dados, o qual representa um arquivo de CRL

- public final CertPath generateCertPath(InputStream fluxoDados)

carrega através de um fluxo de dados, o qual representa um arquivo de uma cadeia de certificados

Nas subseções seguintes serão apresentados os métodos das classes Certificate, X509Certificate e X509CRL. Tais classes tratam dos certificados, fornecendo operações básicas e necessárias para o suporte de implementação de certificados.

4.4.1 Classe Certificate

Esta é uma simples classe abstrata utilizada para gerenciar certificados de entidades. O seu construtor é definido logo abaixo. O tipo do certificado deve ser o X.509. Os métodos que podem ser implementados com um certificado também estão listados abaixo:

- protected Certificate(String tipo)

- public abstract void verify(PublicKey chavePublica)

verifica se o certificado é válido. A chave pública usada deve ser da autoridade certificadora que emitiu o certificado

- public abstract PublicKey getPublicKey()

retorna a chave pública da entidade constante do certificado

- public final String getType()

retorna o tipo do certificado **- public abstract String toString()**

retorna uma representação do certificado

4.4.2 Classe X509Certificate

O tipo, ou seja, o formato do certificado mais comum é o X.509, por isso é o único formato de certificado para o qual existe uma API Java padrão. Com isso faz parte do pacote de segurança a classe X509Certificate.

A classe oferece diversos métodos para obtenção de informações de um certificado. Praticamente todas as propriedades de um certificado são retornadas atráves dos métodos listados abaixo:

- public abstract Date getNotAfter()

retorna a data final do período válido para o certificado

- public abstract Date getNotBefore()

retorna a data inicial do período válido para o certificado

- public abstract BigInteger getSerialNumber()

retorna o número serial do certificado

- public abstract int getVersion()

retorna a versão do certificado

- public abstract String getSigAlgName()

retorna o nome do algoritmo utilizado na assinatura do certificado

Além destes existe o método **checkValidaty()** e **checkValidity(Date data)** que verificam a validade do certificado, sendo o primeiro comparado com a data corrente do computador e o segundo com a data do parâmetro. Não sendo válido pode ocorrer exceções de expirado ou de não válido.

4.4.3 Classe X509CRL

Classe abstrata para uma lista de certificados revogados (CRL). Uma CRL é uma lista com carimbo de tempo que identifica os certificados revogados. A lista é assinada por uma Autoridade Certificadora e disponível em um repositório público.

Cada certificado revogado é identificado em uma CRL por seu número de série. Quando um sistema utiliza assinatura digital e certificados, as verificações

desse sistema não serão somente a assinatura e a validade do certificado, mas também a revogação.

As entradas são adicionadas na CRL enquanto as revogações ocorrem e uma entrada pode ser removida quando a data de expiração do certificado é alcançada.

Abaixo estão os principais métodos desta classe:

- public abstract Date getNextUpdate()

retorna a data da próxima atualização da CRL

- public abstract boolean isRevoked(Certificate certificado)

verifica se o certificado é revogado

4.5 Classe Timestamp

Esta classe possui funcionalidades para se obter o carimbo de tempo. É simples e possui poucos métodos. O carimbo de tempo é obtido com uma data recebida por um servidor de data/hora, e construído com esta data mais o certificado do servidor.

A seguir são listados o método construtor e outros métodos importantes desta classe.

- public Timestamp(Date data, CertPath cadeiaCertif)

construtor da classe

- public Date getTimestamp()

retorna a data constante do carimbo de tempo

- public String toString()

retorna a descrição do carimbo de tempo

4.6 Provedor BouncyCastle

O provedor da BouncyCastle é uma alternativa ao provedor da Sun. Fornece implementações para os serviços de criptografia, contendo recursos que não são fornecidos pela Sun, sendo livre e podendo ser utilizado comercialmente.

A BouncyCastle é um provedor que esta de acordo com a JCE. A vantagem para programar alguma aplicação com este provedor, é que possui funcionalidades extras para a computação criptográfica e ainda pode ser desenvolvida em lugares que a exportação da criptografia é controlada [bou 05].

Com este provedor é possível realizar uma assinatura digital no formato PKCS#7. Na subseção 4.6.1 é realizado o estudo da classe PKCS7SignedData do pacote org.bouncycastle.jce.

4.6.1 Classe PKCS7SignedData

O uso e a verificação das assinaturas digitais é um motor de criptografia padrão que está incluso no pacote jce do provedor BouncyCastle. As implementações das assinaturas digitais são realizadas pela classe PKCS7SignedData.

Para gerar uma assinatura digital de um documento eletrônico, é preciso realizar três etapas que consistem em instanciar um objeto da classe PKCS7SignedData com o método construtor, enviar os dados para serem assinados através do método update e obter o resultado num array de bytes pelo método getEncoded. A sintaxe destes métodos são listados abaixo:

- public PKCS7SignedData(PrivateKey chavePriv, Certificate[] certificados, String algoritmo)

instancia um objeto da classe PKCS7SignerData que utilizará a chave privada e o array de certificados para realizar a assinatura com o algoritmo mencionado

- public void update(byte[] bytes, int posicaoArray, int tamanho)

envia os dados que serão assinados, a partir da posição do array com o tamanho especificado

- public byte[] getEncoded()

obtém o resultado da assinatura digital em um array de bytes

As etapas para a verificação de uma assinatura digital consistem em construir um objeto da classe PKCS7SignedData, receber os dados para verificação através do método update e verificar a assinatura propriamente dita com o método verify. O uso

dos dois primeiros métodos estão descritos na listagem acima, pois são os mesmos tanto para a assinatura, quanto para a verificação, abaixo é listado o único método que é exclusivo para verificar assinaturas digitais:

- public boolean verify()

verifica a assinatura digital

O provedor da BouncyCastle fornece os seguintes algoritmos que podem ser implementados para as assinaturas digitais [bou 05]:

- SHA1withECDSA
- SHA1withDSA
- MD2withRSA
- MD5withRSA
- SHA1withRSA
- RIPEMD128withRSA
- RIPEMD160withRSA
- RIPEMD256withRSA
- SHA224withRSA
- SHA256withRSA
- SHA384withRSA
- SHA512withRSA

4.7 Utilitário Keytool

Nesta seção será visto as funcionalidades do utilitário keytool, o qual acompanha o kit de desenvolvimento Java, permitindo aos seus usuários lidar com os aspectos de segurança e criptografia da linguagem Java, em específico chaves criptográficas e certificados.

Este utilitário fornece funcionalidades através da linha de comando. O keytool permite adicionar, remover e modificar entradas no armazenamento de chaves e certificados.

Existem dois tipos de entradas no armazenamento, as entradas de chave criptográfica, que possui uma chave privada e uma chave pública correspondente. E as entradas de certificado que possui uma chave pública anexada a um certificado.

Todas as entradas no armazenamento são acessadas unicamente através de seu psedônimo. Um pseudônimo é especificado quando adicionado uma entrada no armazenamento usando o comando keytool -genkey para gerar um par de chaves ou o comando keytool -import para adicionar um certificado.

O arquivo de armazenamento é criado durante o uso dos comandos keytool -genkey ou keytool -import, que adicionam dados no arquivo ainda não existente. Existe a opção -keystore para estes comandos que serve para especificar o local do arquivo de armazenamento, se este arquivo não existir, ele é criado, senão a entrada é adiciona nele. Na ausência da opção -keystore, por padrão o arquivo será criado na pasta do usuário do sistema operacional, com o nome .keystore.

Nas demais subseções são examinadas as várias opções de comandos que acompanham o utilitário. Estes comandos estão agrupados por categorias que possuem funcionalidades semelhantes.

4.7.1 Opções Globais

As opções listadas abaixo, podem ser utilizadas com os comandos para adicionar certificado ou gerar par de chaves criptográficas.

-alias pseudonimo especifica o pseudônimo para o comando aplicado

-dname nomeDistinto especifica o nome distinto

-keystore nomeArquivo especifica o nome e local do arquivo de armazenamento

-storepass senha especifica a senha de acesso ao arquivo de armazenamento

O nome distinto deve ser especificado no formato CN=nome comum, OU=unidade da organização, O=organização, L=cidade, S=estado, C=país.

4.7.2 Gerar Chave

Para gerar um par de chaves criptográficas deve-se utilizar o seguinte comando keytool -genkey seguido de diversas opções.

keytool -genkey {-alias pseudonimo} {-keyalg algoritmoChave} {-keysize tamanhoChave} [-dname nomeDistinto] [-keypass senhaChave] {-validity diasValidos} {-keystore localArmazenamento} [-storepass senha]

A opção -keyalg é utilizada para especificar o algoritmo de geração do par de chaves, podendo ser DSA, que é o algoritmo padrão, ou RSA. Se o algoritmo utilizado for DSA, na opção -keysize pode-se especificar um tamanho de 512 a 1024 com números múltiplos de 64, caso contrário deve especificar 1024. Na especificação dos dias válidos para o par de chaves deve-se utilizar -validity, por padrão este valor é 90. A opção -keypass especifica a senha de acesso à chave privada.

4.7.3 Adicionar Certificado

Ao adicionar um certificado deve-se utilizar o seguinte comando keytool -import seguido de algumas opções.

keytool -import {-alias pseudonimo} {-file arquivoCertificado} [-keypass senhaCertificado] {-keystore localArmazenamento} [-storepass senha]

A opção -file indica a localização e o nome do arquivo do certificado a ser importado e -keypass especifica a senha necessária ao acesso do certificado.

4.7.4 Exportar Certificado

O utilitário disponibiliza a possibilidade de exportar certificados para o arquivo de armazenamento usando o comando keytool -export seguido de algumas opções.

keytool -export {-alias pseudonimo} {-file arquivoCertificado} [-keystore localArmazenamento] [-storepass senha]

4.7.5 Deletar Chave ou Certificado

O utilitário não poderia deixar de ter as opções de deletar as entradas no arquivo de armazenamento, cuja finalidade se obtém na utilização do comando keytool -delete seguido de outras opções.

keytool -delete {-alias pseudonimo} [-keystore localArmazenamento] [-storepass senha]

Neste comando é necessário a indicação do pseudônimo da chave ou certificado que se deseja deletar, seguido das opções do local de armazenamento e a senha. Na falta de especificar o nome do pseudônimo, o mesmo será perguntado posteriormente.

4.7.6 Listar Chave ou Certificado

Por fim é apresentado a funcionalidade de listar as entradas armazenadas no arquivo, obtida com o comando keytool -list adicionado de algumas opções.

keytool -list {-alias pseudonimo} [-keystore localArmazenamento] [-storepass senha]

Ao optar por informar o pseudônimo, a ferramenta retornará apenas informações da entrada, ou seja, chave ou certificado, que foi armazenado com aquele pseudônimo. Mas se for omitido a opção -alias, será retornado as informações de todas as entradas.

Capítulo 5

Softwares de Assinatura Digital

5.1 Introdução

A quantidade existente de utilitários para o uso de assinatura digital de um documento eletrônico é grande, existindo softwares simples que utilizam apenas a linha de comando para executar suas tarefas, softwares com funcionalidades estendidas para outro tipo de software, como exemplo um editor de texto, até mesmo os softwares específicos e mais complexos, projetados não apenas para a aplicação da assinatura digital, mas também para realizarem o gerenciamento de certificados e chaves criptográficas, realizarem o cálculo da função resumo e outras funcionalidades, enfim cada software é projetado e desenvolvido com seu diferencial.

Este capítulo tem o objetivo de analisar alguns softwares com o apoio de uma lista de características que são comuns entre os softwares analisados. Inicialmente analisou a licença de uso, com o objetivo de verificar o custo para a aquisição do software. Posteriormente foi necessário verificar os requisitos de instalação, sendo o de suma importância para o estudo, a plataforma exigida, ou seja, o sistema operacional que o software pode ser instalado. Após instalado foi possível obter as funcionalidades que o software proporciona, o seu modo de utilização em termos de acesso do usuário, a estrutura utilizada para o armazenamento de certificados, chaves e lista de contatos e o suporte a token e smart card.

Nas próximas seções estão descritos as características, facilidades e demais informações relevantes dos softwares Cryptonit da Idealx, Assinador do ITI, BRy Signer da BRy Tecnologia, Cert One da Image One, Sistema Omega e o ProSigner da E-Lock.

5.2 Cryptonit

O aplicativo Cryptonit [cry 05] é um software constante de um dos projetos de código aberto conduzido por desenvolvedores da empresa Idealx. É uma ferramenta simples e segura para encriptação e assinatura digital.

Um software desenvolvido para as plataformas Windows e Linux, com sua licença de uso GNU GPL, possuindo as funcionalidades de cifrar e decifrar arquivos, assinar e verificar a validade de assinaturas e ainda a funcionalidade de gerar um arquivo auto-descodificável.

O modo de utilização para que o usuário tenha acesso ao funcionamento do software é multiusuário, onde cada um tem sua própria conta de uso, assim é preservado as suas configurações, os seus contatos e o conjunto de certificados e chaves criptografadas pertencentes a ele.

A estrutura de armazenamento de certificados, chaves e lista de contatos não possui nenhuma proteção de segurança, assim alguém pode deletar tais arquivos, resultando no funcionamento incorreto do software, ou pode trocar os certificados e as chaves, causando um transtorno para o usuário.

A funcionalidade assinar oferece a opção de gerar só a assinatura digital ou a opção de gerar um arquivo com a assinatura e o documento eletrônico, resultando um arquivo com a extensão .pkcs7. O software não permite inserir comentário na assinatura digital.

Já para verificar a assinatura digital o software reconhece as extensões .pkcs7 e .p7s. É possível realizar somente a verificação da assinatura ou verificar e extrair o documento eletrônico.

As funcionalidades de cifrar e decifrar um arquivo são simples, sem

nenhum ponto inportante a ser relatado. Por fim a função de gerar um arquivo auto-descodificável é definida numa função que cifra um arquivo com uma senha e tem como resultado um arquivo executável, que ao ser executado solicita a senha para ser decifrado.

Para o usuário utilizar sua chave privada e seu certificado, é necessário armazená-los junto ao software, para isso é preciso importar de um arquivo que esteja no formato PKCS#12 ou de uma requisição de certificado. Contudo o software não possui suporte a token e smart card.

Existe algumas facilidades que são importantes de serem informadas, como o gerenciamento de certificados que o software realiza, importando certificados em diferentes formatos, os quais devem ser arquivos com as extensões .cer, .der ou .pem. O software gera arquivo contendo requisição de certificado e ainda tem a opção de importar a lista de certificados revogados de um arquivo, de uma URL ou de um servidor LDAP.

5.3 Assinador ITI

O Instituto Nacional de Tecnologia da Informação, órgão vinculado à Casa Civil da Presidência da República, promoveu o software Assinador [ITI 05]. E para a implementação deste aplicativo, formou-se a parceria de desenvolvimento da E-Sec Tecnologia em Segurança de Dados com a CertiSign Certificadora Digital, que aproveitaram os códigos-fonte do aplicativo Cryptonit, software analisado na seção 5.2.

A licença de software associada ao aplicativo Assinador é a GNU GPL, ou seja, software de código aberto que é disponibilizado pelo ITI para instalação, utilização e modificação.

O Assinador, semelhante ao Cryptonit, é um software com uma interface gráfica, porém de uso apenas no sistema operacional Linux. Tem sua finalidade no uso para a certificação digital, em específico assinar e verificar documentos eletrônicos e cifrar e decifrar arquivos.

Diferente do Cryptonit, neste software pode-se utilizar os certificados da ICP-Brasil tipo A3, ou seja, aqueles que estão armazenados em um token ou smart card.

O modo de utilização para que o usuário tenha acesso ao funcionamento do software, a estrutura de armazenamento de certificados, chaves e lista de contatos e as funcionalidades de assinar, verificar cifrar e decifrar documentos eletrônicos são idênticas ao software Cryptonit.

As facilidades também são idênticas ao software Cryptonit, exceto o gerenciamento de certificados do Assinador que aceita também a importação de outro tipo de certificado, o qual esteja no formato de arquivo com a extensão .crt.

5.4 BRy Signer

O BRy Signer [bry 05] é um software de assinatura digital desenvolvido pela BRy Tecnologia, empresa nacional que desenvolve, produz e comercializa produtos e serviços relacionados com a segurança da informação.

É um software desenvolvido para a plataforma Windows que pode ser instalado nos sistemas operacionais Windows 9x, ME, 2000, NT e XP e ainda requer o navegador Internet Explorer 5.0 ou superior. Sua licença de uso é proprietária, mas disponibiliza uma versão de demonstração pelo período de 60 dias. Posui as funcionalidades de assinar e verificar a validade de assinaturas digitais, co-assinar arquivos já assinados e ainda abrir um arquivo contido em uma assinatura digital no formato PKCS#7.

O modo de utilização para que o usuário tenha acesso ao funcionamento do software, às chaves privadas e aos certificados é definido no perfil de login do sistema operacional, ou seja, dependendo de quem logou, o software terá acesso ao conjunto de chaves privadas e certificados armazenados configurados para aquele perfil.

A estrutura de armazenamento de certificados e chaves privadas é oculta pelo sistema operacional, a qual é armazenada em arquivos e diretórios mascarados.

A funcionalidade assinar gera um arquivo, no formato PKCS#7 e com a extensão .p7s, que contém a assinatura digital e o documento eletrônico. Assina diversos arquivos numa mesma operação, gerando arquivos correspondentes para cada arquivo assinado. O software permite inserir comentário na assinatura digital. Da mesma forma a funcionalidade co-assinar realiza as mesmas funções da funcionalidade assinar, só que é

exigido que seja um arquivo resultado de assinatura digital.

A verificação da assinatura digital pode ser feita com arquivos de extensões .p7s, .sig e .brysig. E ainda oferece a opção de salvar o documento eletrônico, contido no arquivo de assinatura digital, em um diretório local.

Para o usuário utilizar sua chave privada e seu certificado, é necessário armazená-los através do gerenciador de certificados do sistema operacional Windows. Assim o software possui suporte a token e smart card, desde que os mesmos estejam configurados no sistema operacional.

O software não gera arquivo contendo requisição de certificado e ainda tem a opção de importar a lista de certificados revogados de uma URL constante nas informações do certificado.

5.5 Cert One

O Cert One [cer 05] é o software de certificação digital da Image One. A integridade dos arquivos assinados eletronicamente gerados por este software é garantida através de uma função resumo de segurança própria, criada a partir da combinação das informações do arquivo juntamente com a chave privada do assinante.

Software desenvolvido para a plataforma Windows, possui sua licença de uso proprietária, porém disponibiliza uma versão completa de demonstração pelo período de 10 dias e uma outra versão por período indeterminado, a qual apenas verifica a assinatura digital. Posuindo as funcionalidades de assinar e verificar a validade de assinaturas digitais e abrir um arquivo contido em uma assinatura digital no formato próprio do software.

A estrutura de armazenamento de certificados e chaves privadas, o modo de utilização do usuário ao acesso do funcionamento do software e o gerenciamento de certificados, são funcionalidades semelhantes ao software analisado na seção 5.4, sendo dependentes da configuração do sistema operacional.

Ao assinar um documento eletrônico, é gerado um arquivo com a extensão .not, que contém a assinatura digital e o documento eletrônico, podendo ainda

conter um comentário, pois o software permite a inserção de comentários na assinatura digital. O software assina apenas um arquivo por operação. Como o software gera um arquivo de assinatura digital num formato próprio, a verificação desta assinatura somente pode ser feita com arquivos de extensão .not.

Não possui a opção de gerar arquivo contendo requisição de certificado e também não importa ou atualiza a lista de certificados revogados de alguma fonte qualquer.

5.6 Sistema Omega

O Sistema Omega [ome 05] é um software idealizado e desenvolvido para a plataforma Windows, possui sua licença de uso proprietária, porém atualmente é disponibilizada uma versão funcional que não possui data de expiração, contudo está limitado ao uso de apenas dois usuários.

Posui as funcionalidades de assinar e verificar a validade de assinaturas digitais. Na função de assinar um documento eletrônico, gera um arquivo no formato PKCS#7 com a extensão .p7m, assina apenas um arquivo por vez e não permite inserir comentário na assinatura. Para a verificação reconhece arquivos assinados na extensão .p7m.

Também possui as funcionalidades de cifrar e decifrar arquivos. Cifra um arquivo com a chave pública, constante de um certificado do destinatário, tal certificado é reconhecido nas extensões .crt e .cer. O resultado da função cifrar é um arquivo na extensão .p7a.

O modo de acesso às funcionalidades do software é baseado na escolha de uma chave privada. Tal chave é protegida por senha, a qual valida o acesso ao usuário. A estrutura de armazenamento de certificados, chaves e lista de contatos não possui proteção de segurança, localizada no diretório Omega, onde alguém pode deletar tais arquivos, resultando no funcionamento incorreto do software.

Para o usuário utilizar seu certificado, é necessário armazená-lo junto ao software, o qual aceita importar arquivos com as extensões .crt e .cer. Contudo o software

não possui suporte a token e smart card.

5.7 ProSigner

O ProSigner [elo 05] é um software de assinatura digital desenvolvido pela E-Lock para a plataforma Windows, tendo suas funcionalidades estendidas para aplicativos da Microsoft, como o Word e o Excel.

Sua licença de uso é proprietária, mas é disponibilizado uma versão de demonstração pelo período de 30 dias. Possui as funcionalidades de assinar e verificar a validade de assinaturas digitais e cifrar e decifrar arquivos.

O modo de utilização para que o usuário tenha acesso ao funcionamento do software, a estrutura de armazenamento de certificados e chaves privadas e o suporte a token e smart card, são semelhantes ao BRy Signer, software analisado na seção 5.4.

A funcionalidade assinar gera um arquivo, no formato PKCS#7 com a extensão .p7m. Assina um arquivo por operação e não permite inserir comentário na assinatura digital. A verificação da assinatura digital pode ser feita com arquivos de extensões .p7s, .pkcs7 e .p7m. E durante a verificação da assinatura, o software acessa a URL para validação do certificado, consultando a lista de certificados revogados.

Para que o usuário possa utilizar sua chave privada e seu certificado, é necessário armazená-los através do gerenciador de certificados do sistema operacional.

O software não gera arquivo contendo requisição de certificado e ainda tem a opção de importar ou atualizar a lista de certificados revogados a partir de um arquivo.

5.8 Conclusão

Com o objetivo deste capítulo foi possível analisar as funcionalidades de alguns softwares, verificar os pontos positivos apresentados e também os pontos negativos e por fim verificar se os arquivos assinados podem ser interoperáveis.

Existem funcionalidades semelhantes em quase todos os softwares estudados. A diferença de um para o outro, é a facilidade apresentada para uma determinada atividade do usuário. Mas também existe funcionalidade que só um ou outro software disponibiliza.

Positivamente foi observado a facilidade de uso que a maioria dos softwares apresentam, incluindo a possibilidade de estender as funcionalidades para outros aplicativos. Outro fator positivo foi os padrões dos arquivos assinados, ou seja, no formato PKCS#7, em que é presente na maioria dos softwares analisados. Contudo nem todos os softwares são operáveis entre eles, devido ao formato dos seus arquivos assinados, o que foi observado como um ponto negativo.

O ponto negativo de maior expressão é o custo para aquisição do software que exige licença de uso para o seu funcionamento. Num total de seis softwares analisados, quatro são de licença proprietária, resultando em mais de 66%.

Ter como requisito de instalação apenas o sistema operacional Windows, também foi verificado como um ponto negativo. O que torna o software dependente de plataforma e deixa o usuário de outras plataformas, em especial o Linux, sem a possibilidade de utilizar o software.

Outro fator observado é a dependência de alguns softwares com o sistema operacional, para as funcionalidades de gerenciamento de certificados e importação de chaves privadas.

Por fim alguns softwares possuem uma funcionalidade que é observado com supérflua. A função de gerar uma requisição de certificado para uma AC. Já que para a ICP-Brasil é preciso que o solicitante de um certificado compareça na AR para apresentar alguns documentos.

Capítulo 6

Proposta do Software

6.1 Introdução

A proposta do software para assinatura digital, objetivo geral deste projeto, tem sua base definida a partir da análise dos softwares existentes de assinatura digital, a qual resultou na apresentação do capítulo 5.

A partir daquela análise é determinado o escopo do software, que consequentemente define os requisitos necessários para o desenvolvimento do software.

Na seção 6.2 é apresentado a proposta com as definições das suas funcionalidades, em seguida na seção 6.3 é listado os requisitos que satisfazem a proposta apresentada.

6.2 Proposta

A proposta do software está definida na figura 6.1, inserida na página 44. Seus propósitos são dispor funcionalidades que foram pontos positivos dos softwares analisados, apresentar uma implementação diferente daquelas funcionalidades que formaram os pontos negativos e deixar de implementar as funcionalidades julgadas como supérfluas.

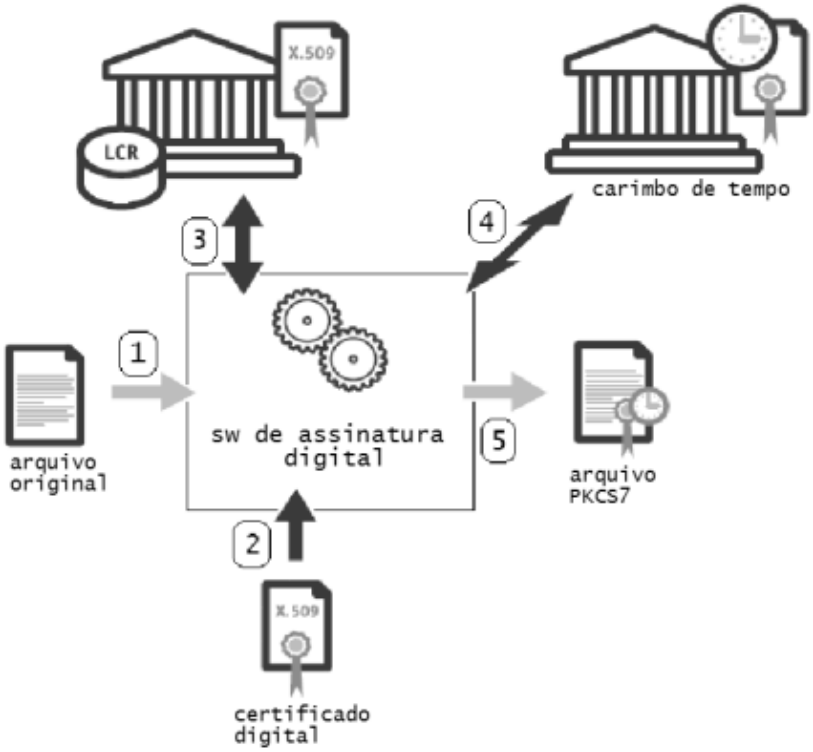

Figura 6.1: Proposta do Software para Assinatura Digital

Na figura 6.1 é demonstrado as atividades do usuário ou as funcionalidades do software através dos números impressos nela. Inicialmente o usuário escolhe o arquivo original que deseja ser assinado digitalmente. Após a escolha, o software lista os certificados do usuário que estão armazenados na keystore. Em consequência o software faz consulta à CRL do certificado para verificar a revogação. Não havendo restrições ao uso do certificado, o software requisita o carimbo de tempo a um servidor de data/hora. Finalizando apresenta o resultado da assinatura digital no formato PKCS#7.

6.3 Requisitos

Diante do escopo definido na seção 6.2 é possível analisar os requisitos necessários para o desenvolvimento do software com o objetivo de apresentar um aplicativo com as funcionalidades desejadas.

O software possui funcionalidades básicas, conforme descrito em uma lista de requisitos para a implementação do aplicativo:

- autenticação
- assinatura de documento eletrônico
- verificação de assinatura
- gerenciamento de chaves públicas e privadas
- gerenciamento de certificados digitais

Após identificar as necessidades para o desenvolvimento do software, através da análise de requisitos, é necessário apresentar de forma clara as funcionalidades que fazem parte do sistema, com objetivo de facilitar sua compreensão, possíveis melhorias nas suas funcionalidades e futuras manutenções no aplicativo. Para tanto se utiliza de descrições de casos de uso, adicionados de relações de funções e atributos do sistema.

As descrições dos casos de uso ajudam no entendimento do software, apontando o autor e a descrição sumária para uma atividade do sistema. Já as relações de funções do sistema especificam as atividades do sistema de forma implícita, pois não são aparentes para o usuário. E as relações de atributos do sistema detalham as qualidades não-funcionais do sistema, tais como facilidades de uso [LAR 00].

6.3.1 Casos de Uso

As relações a seguir descrevem de maneira detalhada cada uma das atividades que podem ser exercidas pelo usuário e auxiliam no melhor entendimento do funcionamento do software. Estas atividades praticamente representam as funcionalidades disponíveis do software para assinatura digital.

CASO DE USO AUTENTICAR
ATOR usuário
DESCRIÇÃO começa quando o software é inicializado, o usuário deve informar

seu login e senha e após validação terá acesso às funcionalidades do aplicativo.

CASO DE USO CRIAR USUÁRIO
ATOR usuário
DESCRIÇÃO começa quando o software é inicializado e o usuário opta em criar um novo usuário para o aplicativo, ele deve preencher os dados com o nome delogin e a senha.

CASO DE USO ASSINAR DOCUMENTO
ATOR usuário
DESCRIÇÃO inicia quando usuário optar por assinar um documento eletrônico, ele deve indicar o local onde esta armazenado o arquivo e se houver mais de uma chave privada cadastrada para seu uso, deve escolher qual utilizar para a realização da assinatura.

CASO DE USO VERIFICAR ASSINATURA
ATOR usuário
DESCRIÇÃO inicia quando usuário optar por verificar a assinatura de um documento eletrônico, ele deve indicar o local onde se encontra o arquivo de assinatura digital.

CASO DE USO ADICIONAR CERTIFICADO OU PAR DE CHAVES
ATOR usuário
DESCRIÇÃO inicia quando usuário optar por armazenar um certificado de confiança ou um par de chaves criptográficas na sua keystore.

CASO DE USO REMOVER CERTIFICADO OU PAR DE CHAVES
ATOR usuário
DESCRIÇÃO inicia quando usuário optar por remover um certificado de confiança ou um par de chaves criptográficas da sua keystore.

6.3.2 Funções de Sistema

Nesta subseção serão descritos as funções de sistema necessárias para que os casos de uso sejam executados, em conseqüência atendendo os requisitos do software.

Existe uma função de sistema para a validação de acesso ao usuário, que consiste em verificar a existência de um login informado e na existência deste, verifica se a senha é correspondente. Esta função é importante para o usuário obter acesso às funcionalidades do aplicativo.

No caso de uso criar usuário, o sistema deve criar um diretório e gerar uma keystore para o usuário que está sendo cadastrado. Assim será solicitado algumas informações ao usuário necessárias para a concretização da tarefa.

O resultado do caso de uso assinar documento é um arquivo de assinatura digital e um arquivo de carimbo de tempo, para a realização desta tarefa existe a função de sistema gerar tais arquivos. Que consiste em salvar no mesmo diretório do arquivo assinado, o arquivo resultante de assinatura digital e o arquivo de carimbo de tempo.

No mesmo caso de uso assinar documento existe uma função que detalha a chave utilizada pelo usuário para a assinatura digital. Esta atividade serve para mostrar informações da chave privada.

Ainda no caso de uso assinar documento, o software tem a função de consultar a CRL da AC que emitiu o certificado que esta sendo utilizado para a assinatura digital. Para tanto verifica a URL da CRL, faz a requisição e verifica se o certificado não consta na lista.

Finalizando o caso de uso assinar documento, o software envia uma requisição através do protocolo SNTP para um servidor de data/hora, para obter a hora oficial e construir o carimbo de tempo.

Quando o usuário solicita a verificação de uma assinatura digital, é preciso que uma função de sistema complete a funcionalidade do aplicativo. A função consiste em verificar a assinatura digital do documento eletrônico, retornando uma mensagem

informativa, conforme a situação da verificação. Consulta também o arquivo de carimbo de tempo, para obter a data e comparar com a data de modificação do arquivo original. E ainda consulta a CRL para verificar a revogação do certificado.

Na adição de um certificado ou par de chaves, a função de sistema possibilita ao usuário a importação de certificado contido em um arquivo. Após a seleção deste arquivo o sistema solicita informações ao usuário referente ao certificado que será adicionado.

Já na remoção de um certificado ou par de chaves, a função de sistema permiti que o usuário remova um certificado armazenado na keystore.

6.3.3 Atributos do Sistema

Esta subseção tem o propósito de demonstrar os atributos de sistema adotados para o desenvolvimento do software para assinatura digital. Os atributos são características ou dimensões do sistema e por serem qualidades não-funcionais, não dependem ou influenciam nas funcionalidades do aplicativo, nem atendem os requisitos exigidos.

Os atributos determinados para o software especificam uma interface simples com informações necessárias ao usuário, sendo prioridade sua facilidade de uso.

O ambiente principal de utilização do software é uma interface gráfica, composta exclusivamente de botões, que possibilita o acesso direto às funcionalidades do software. Tais botões possuem nome simples e são de fácil entendimento, indicando a função que será desempenhada ao ser clicado.

Capítulo 7

Funcionalidades do Software

7.1 Introdução

Neste capítulo serão apresentadas as funcionalidades do software através de figuras que representam as interfaces do aplicativo. Também será comentado algumas implementações realizadas no código-fonte.

7.2 Autenticação

Ao inicializar o software para assinatura digital o usuário interage com uma interface de validação de acesso. A partir desta pode ter acesso à interface principal ou à interface para criação de usuário.

A verificação da senha do usuário é realizada ao carregar uma keystore. Se a senha estiver correta a keystore é carregada e o usuário tem acesso à interface principal. Isto é implementado no método logar da classe DialogoLogin, código-fonte listado na seção B.15, onde existe uma chamada ao método carregaKeystore.

```
...
72        try {
73           if (!keystore.carregaKeystore(usuario.informaSenha())) {
74              JOptionPane.showMessageDialog(this, "senha incorreta",
                 "erro", JOptionPane.ERROR_MESSAGE);
75              limparCampos();
```

```
76           } else {
77              janelaPai.recebaObjs(keystore, usuario);
78              janelaPai.setTitle("assinatura digital - " + log);
79              setVisible(false);
80           }
...
```

Figura 7.1: Interface Validação de Acesso

A figura 7.1 é a interface na qual o usuário interage para validar o seu acesso, tendo os campos de login e senha. E ainda a opção para criar um novo usuário do aplicativo.

7.3 Criação de Usuário

Esta funcionalidade esta implementada na classe DialogoCriarUsuario, através do método criar, código-fonte constante na seção B.14. Um trecho deste código é listado abaixo. Ao criar um novo usuário, é chamado o método criarDiretorio, linha 68, o qual cria um diretório e uma nova keystore. Se o nome informado já existir é mostrado uma mensagem, conforme as linhas 66 e 67.

```
...
66        if (keystore.existe()) {
67           JOptionPane.showMessageDialog(this, "usuário já existe",
              "erro", JOptionPane.ERROR_MESSAGE);
68        } else if (criarDiretorio(log)) {
69           try {
```

```
70              keystore.criaKeystore(usuario.informaSenha());
71           } catch (Exception e) {
72              e.printStackTrace();
73           }
...
```

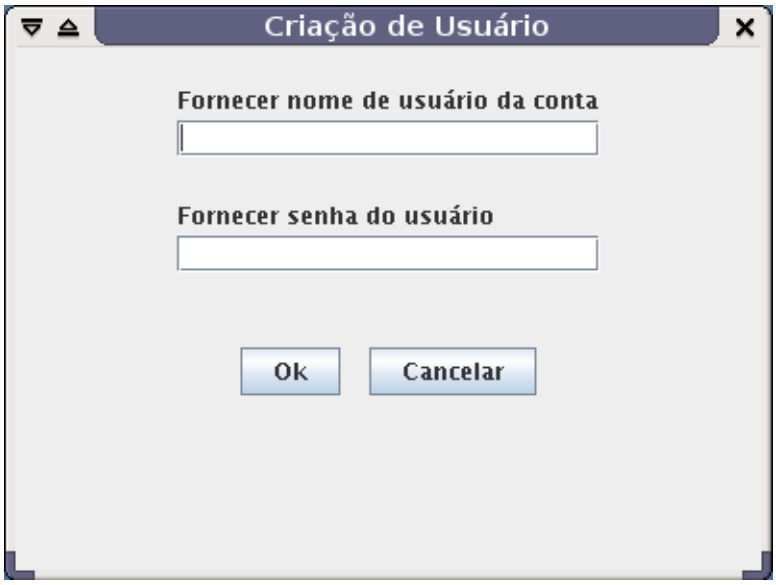

Figura 7.2: Interface Criação de Usuário

A figura 7.2 é a interface para criar um novo usuário para o software, consta de um campo para fornecer um nome e outro campo para fornecer a senha.

7.4 Assinatura

A funcionalidade de assinar um arquivo é acessada ao clicar o botão correspondente da interface principal. As etapas para assinar um arquivo estão listadas abaixo, sendo que as duas primeiras necessitam da interação do usuário e as demais são realizadas automaticamente:

- Selecionar um arquivo;
- Escolher a chave privada;

- Definir o algoritmo da função resumo;
- Verificar a validade do certificado;
- Obter a CRL do certificado;
- Obter o carimbo de tempo;
- Realizar a assinatura;
- Salvar o arquivo resultado da assinatura;
- Salvar o arquivo do carimbo de tempo.

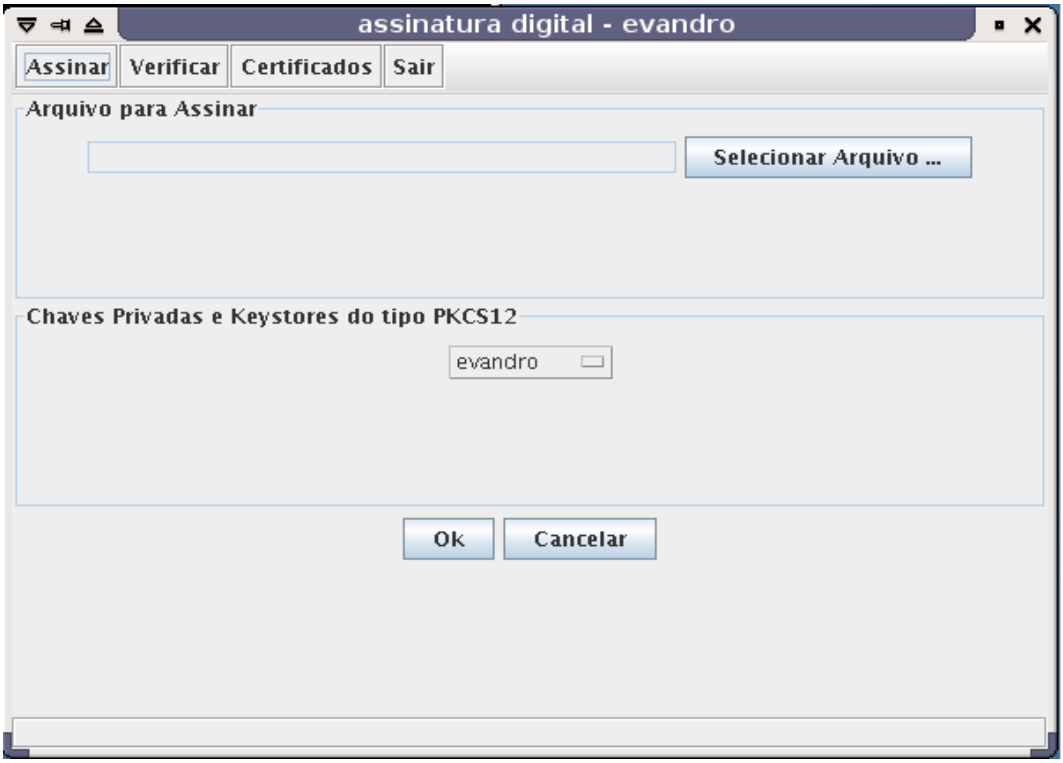

Figura 7.3: Interface Assinar

A figura 7.3 representa a interface para o usuário assinar um arquivo. Ele deve selecionar um arquivo e escolher uma chave privada, de uma lista de chaves armazenadas na sua keystore padrão ou nas keystores do tipo PKCS#12.

A lista das chaves privadas do usuário é construída através do método mostrarChaves da classe PainelAssina, a qual é implementada na classe JanelaPrincipal, listada na seção B.17. Além da keystore padrão “keystore.bks”, outras keystores do diretório são adicionadas na lista, como a implementação da linha 167.

```
...
160       protected void mostrarChaves() {
161          campoChave.removeAll();
162          Vector vetor = new Vector();
163          try {
164             if (usuario.informaKeystores() == null) {
165                vetor = keystore.listaChave();
166             } else {
167                vetor = keystore.listaChaveDiversas(usuario.informaKeystores());
168             }
...
```

O algoritmo da função resumo é determinado conforme o algoritmo da chave privada, se for RSA o algoritmo da função resumo será MD5, se for DSA será SHA.

Na validade do certificado é requisitado a data e a hora atual a um servidor de tempo, conforme o método informaValidade da classe Certificado, seção B.12, listado logo abaixo. Pode-se observar na linha 61 a instância do objeto sntp, tendo como parâmetro o IP do Observatório Nacional. As classes SntpClient, seção B.21, e NtpMessage, seção B.19, fazem parte da implementação de um cliente NTP[1] em Java [ntp 06]. Se não houver acesso a internet, o método utiliza a data e a hora do computador.

```
...
58    public boolean informaValidade(Certificate certif) {
59       Date data;
60       try {
61          SntpClient sntp = new SntpClient("200.20.186.75");
62          data = sntp.informaData();
63       } catch (IOException io) {
64          System.out.println("AVISO - sem acesso a internet");
65          data = new Date();
66          System.out.println("Utilizando a hora do computador\n");
67       }
...
```

[1]NTP é um protocolo para sincronizar relógios de computadores através de uma rede.

Caso o certificado seja válido, a próxima etapa é a obtenção da CRL, para verificar se o mesmo não esta revogado. E após isto é obtido o carimbo de tempo.

A assinatura é realizada com um array de bytes, obtidos do arquivo original e também com o array de bytes da data obtida do carimbo de tempo. O método assinar da classe PainelAssina, linha 303, invoca uma chamada ao método assinar da classe Assina, seção B.10, conforme pode-se observar no trecho listado abaixo.

```
...
302             Assina assina = new Assina();
303             if (assina.assinar(priv, certifs, arq, data, algo)) {
304                carimbo.salvar(arq);
305                JOptionPane.showMessageDialog(this,
                    ("resultado da assinatura\n"
                     + arqAssinar.getPath() + ".pkcs7"), null,
                      JOptionPane.INFORMATION_MESSAGE);
306             } else {
307                JOptionPane.showMessageDialog(this,
                    "assinatura não realizada", "erro",
                     JOptionPane.ERROR_MESSAGE);
...
```

Por fim é salvo em um arquivo, o resultado da assinatura com o nome do arquivo original seguido da extensão “.pkcs7” e em outro arquivo, o carimbo de tempo seguido da extensão “.stamp”.

7.5 Verificação

Para verificar um arquivo assinado digitalmente é necessário clicar no botão da interface principal. As etapas para verificar um arquivo estão listadas abaixo, sendo que apenas a primeira necessita da interação do usuário e as outras são realizadas automaticamente:

- Selecionar um arquivo;
- Obter o arquivo original;
- Obter o arquivo do carimbo de tempo;

- Comparar a data do arquivo original com a data do carimbo de tempo;
- Verificar a assinatura.

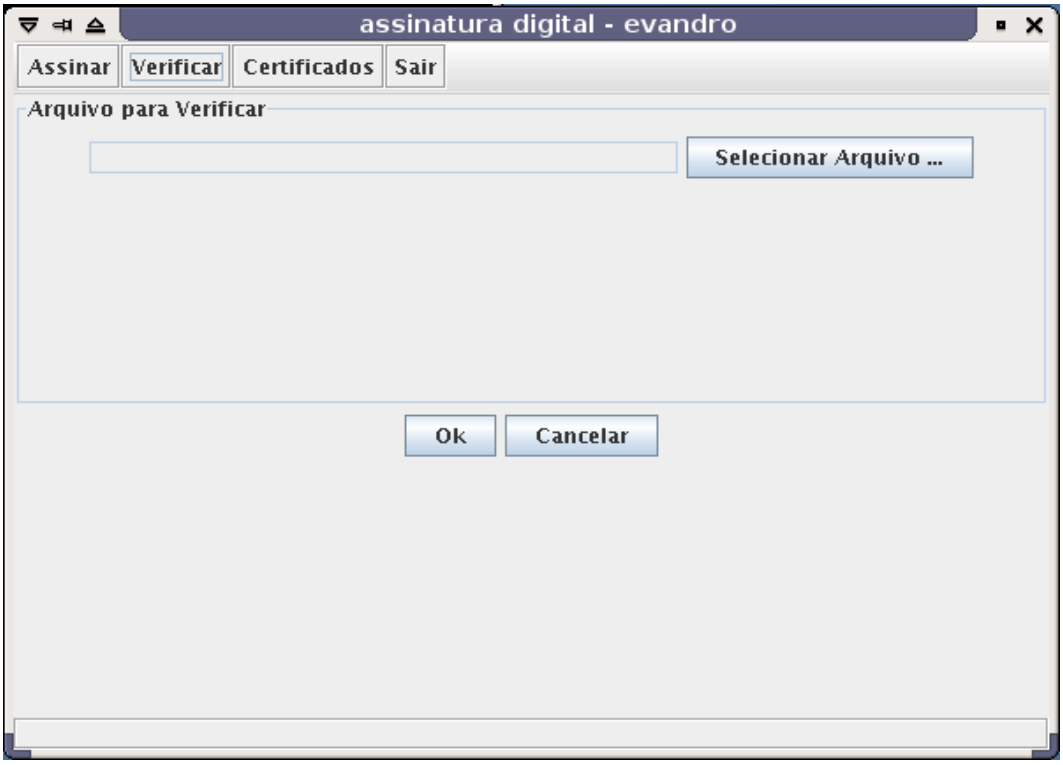

Figura 7.4: Interface Verificar

A figura 7.4 representa a interface para o usuário verificar um arquivo assinado digitalmente. Nela seleciona-se o arquivo resultado de uma assinatura, sendo que o arquivo carimbo de tempo e o arquivo original devem estar no mesmo diretório do arquivo resultado.

O arquivo original e o arquivo do carimbo de tempo são obtidos pelo método verificar da classe Verifica, seção B.22, o qual é invocado pelo método verificar da classe PainelVerifica, um trecho do método da classe PainelVerifica é listado a seguir.

```
...
383          Verifica verifica = new Verifica();
384          try {
385             if (verifica.verificar(arqVerificar.getPath())) {
```

```
386                JOptionPane.showMessageDialog(this,
                    "verificação de assinatura correta",
                     null, JOptionPane.INFORMATION_MESSAGE);
387             } else {
388                JOptionPane.showMessageDialog(this,
                    "verificação de assinatura incorreta",
                     "erro", JOptionPane.ERROR_MESSAGE);
389             }
...
```

Ao comparar as datas, é verificado se a data do carimbo de tempo é anterior à data do arquivo original, caso seja, a verificação é incorreta. Após estas etapas é verificado a assinatura utilizando o método verificar da classe Verifica, seção B.22.

7.6 Gerenciamento de Certificados

O gerenciamento de certificados é obtido ao clicar no botão Certificados da interface principal.

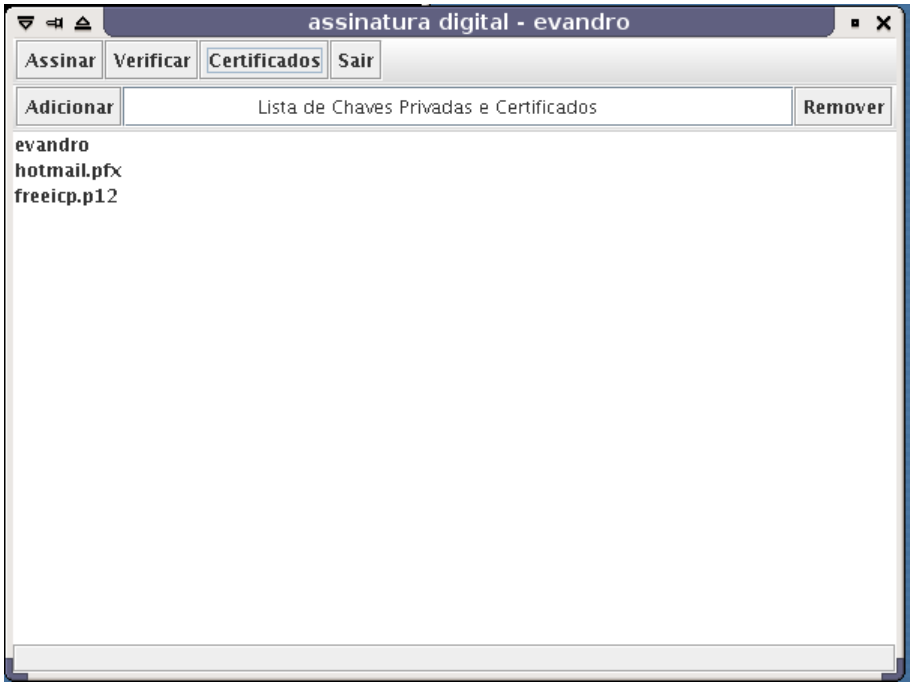

Figura 7.5: Interface Certificados

A figura 7.5 apresentada abaixo, possui o botão Adicionar para inserir certificados na keystore padrão ou salvar keystores do tipo PKCS#12 no diretório de trabalho do usuário. Já com o botão Remover exclui certificados ou deleta keystores.

O botão adicionar é implementado com o método adicionaItemLista da classe PainelCertif, a qual é interna à classe JanelaPrincipal. Conforme a lista abaixo, na linha 455 existe uma decisão para a seleção do tipo de certificado que se deseja armazenar, caso o certificado esteja em um arquivo .cer ou .crt, ele será armazenado na keystore, caso esteja num arquivo .p12 ou .pfx, será salvo no diretório de trabalho do usuário.

```
...
455             if (diretorio.salvarCertif(arqCertif, usuario.informaDirKeys())) {
456                JOptionPane.showMessageDialog(this, "arquivo "
                    + arqCertif.getName() + " salvo", null,
                     JOptionPane.INFORMATION_MESSAGE);
457                this.informaLista();
458             } else {
459                alias = JOptionPane.showInputDialog(this,
                    "Digite um nome", "Alias do Certificado",
                     JOptionPane.PLAIN_MESSAGE);
460                try {
461                   Certificado certificado = new
462                   Certificado(arqCertif, alias, keystore,
463                   usuario.informaSenha());
464                   if (keystore.existeEntrada(alias)) {
465                      this.informaLista();
466                      JOptionPane.showMessageDialog(this,
                          "certificado " + alias + " adicionado",
                           null, JOptionPane.INFORMATION_MESSAGE);
...
```

Ao remover um certificado é necessário selecionar um item da lista de certificados, atráves do método removeItemLista da classe PainelCertif. No trecho abaixo, na linha 484 pode-se notar a implementação para a remoção de uma keystore do tipo PKCS#12 e na linha 487 nota-se a remoção de um certificado armazenado na keystore padrão.

```
...
484                if (diretorio.removerCertif(alias,usuario.informaDirKeys())) {
485                   JOptionPane.showMessageDialog(this,"arquivo removido",
```

```
                    null, JOptionPane.INFORMATION_MESSAGE);
486             } else {
487                 keystore.deletaEntrada(alias, usuario.informaSenha());
488                 JOptionPane.showMessageDialog(this, "certificado removido",
                    null, JOptionPane.INFORMATION_MESSAGE);
...
```

Capítulo 8

Considerações Finais

8.1 Conclusões

O desenvolvimento do software para assinatura digital teve três linhas de pesquisa, a primeira baseou-se na criptografia e posteriormente o estudo foi em torno da linguagem de programação Java, finalizando com a análise de softwares de assinatura digital.

O estudo da criptografia foi necessário, essencial e de fundamental importância para o embasamento teórico do desenvolvimento do software, que resultou no conhecimento de seus conceitos e de suas funcionalidades. Foi relatado as diferenças da criptografia simétrica para a assimétrica e demonstrado os mais importantes algoritmos utilizados para cifrar e decifrar arquivos. Foi preciso também entender a função resumo e a sua finalidade para a assinatura digital. Por fim foi visto os principais algoritmos para gerar e verificar assinaturas digitais. Este conjunto de estudos fizeram parte de um dos objetivos específicos do projeto, sendo relatado no capítulo 2.

Em continuidade ao estudo da criptografia, mas focalizando-o ao objetivo geral do projeto, foi visto no capítulo 3 um detalhamento sobre a assinatura digital. Estudou-se as suas diversas características que trazem vantagens para o seu uso. O seu funcionamento, tanto para assinar como para verificar um documento eletrônico, foi mostrado etapa por etapa com a finalidade de um melhor entendimento. E ainda a respeito

da assinatura digital, foi mostrado as suas possíveis aplicabilidades seguidas da legislação pertinente que a torna válida juridicamente.

O capítulo 3 ainda constou um breve relato sobre o certificado digital e sobre a infra-estrutura de chaves públicas brasileira, pois estão diretamente relacionados à assinatura digital. O certificado porque consta de informações sobre o assinante de um documento eletrônico recebido e principalmente da sua chave pública. E a ICP-Brasil porque é uma cadeia de instituições públicas ou privadas, responsáveis pela integridade, autenticidade e validade jurídica dos certificados e chaves criptográficas.

A segunda linha de pesquisa concentrou no capítulo 4, com o estudo das classes da linguagem de programação Java, especificamente API's de segurança. A importância deste estudo esteve relacionada à construção propriamente dita do software, servindo de embasamento prático, pois através da utilização das classes de segurança Java, obteve o resultado de uma implementação mais eficiente e otimizada do código fonte.

Neste capítulo 4 foi visto a arquitetura de criptografia Java, que possui classes projetadas para permitir funcionalidades de nível de segurança para o desenvolvimento de programas. Foram relatados métodos das classes que implementam a keystore, os certificados digitais, o carimbo de tempo e a assinatura digital do provedor da Bouncy Castle. E também foi visto um utilitário de linha de comando aplicado ao gerenciamento do armazenamento de chaves e certificados digitais.

A pesquisa de funcionalidades de um aplicativo de assinatura digital, foi realizada no capítulo 5. Este estudo foi realizado com a instalação do software para observar o seu funcionamento. Assim foi possível verificar as suas características, a sua forma de funcionamento, as facilidades e constatar a interoperabilidade entre os arquivos resultantes em função de assinar um documento eletrônico.

O capítulo 6 fez parte da apresentação de uma proposta para um software de assinatura digital, utilizando as funcionalidades analisadas no capítulo 5 e implementando novas funcionalidades. Foi definido o escopo do software e a partir desta etapa, foram descobertos os requisitos necessários que satisfazem as exigências do aplicativo. Com os requisitos definidos, obteve-se os casos de uso, os quais foram detalhados. Finalizando com as descrições das funções e dos atributos do sistema.

Por fim utilizou-se do capítulo 7 para demonstrar as funcionalidades e as implementações do software para assinatura digital, onde foram apresentadas as interfaces do software e analisadas algumas linhas do código-fonte.

8.2 Trabalhos Futuros

O desenvolvimento de trabalhos futuros esta relacionado com o objetivo de aperfeiçoar a implementação da proposta de software apresentada. Um ponto de necessidade é aumentar as suas funcionalidades em termos de segurança, adicionando a possibilidade de assinar documentos através de certificados e chaves criptográficas armazenados num dispositivo removível, como um smart card ou token.

A primordial sugestão que aqui se descreve tem relação com o formato do arquivo resultado da assinatura digital. É importante que ocorra a interoperabilidade entre os softwares existentes, portanto a necessidade de implementar o arquivo no formato PKCS#7, com o arquivo original, com o arquivo de assinatura digital e com arquivo do carimbo de tempo, é um trabalho a ser desenvolvido.

Outra implementação de suma importância é o ajuste do software para aceitar URL de uma lista de certificados revogados que esteja no protocolo https. Pois não tendo uma CRL, o software não consulta a revogação do certificado.

Referências Bibliográficas

[bou 05] **The Legion of the Bouncy Castle**. Disponível em <http://www.bouncycastle.org>. Acesso em: 03 de dezembro de 2005.

[bry 05] **BRy Signer**. Disponível em <http://www.bry.com.br/downloads/produtos.asp>. Acesso em: 08 de novembro de 2005.

[BUA 03] BUARQUE, V. J. Padrão de criptografia simétrica - DES/AES. [S.l.], 07 de março de 2005, 2003.

[BUR 02] BURNETT, S.; PAINE, S. **Criptografia e Segurança: O guia oficial RSA**. Rio de Janeiro: Editora Campus, 2002.

[cer 05] **Assinador - Cert One**. Disponível em <http://www.assinador.com.br/certone.asp>. Acesso em: 10 de novembro de 2005.

[COS 03] COSTA, V. et al. Protocolação digital de documentos eletrônicos. In: I FÓRUM SOBRE SEGURANÇA, PRIVACIDADE E CERTIFICAÇÃO DIGITAL, 2003. [s.n.], 2003.

[cry 05] **Cryptonit**. Disponível em <http://cryptonit.org/index.en.html>. Acesso em: 22 de outubro de 2005.

[CUS 03] CUSTÓDIO, R. F.; DIAS, J. S.; ROLT, C. R. D. Assinatura confiável de documentos eletrônicos. [S.l.], 08 de outubro de 2004, 2003.

[DEI 01] DEITEL, H. M.; DEITEL, P. J. **Java Como Programar - 3ª Edição**. Porto Alegre: Bookman, 2001.

[dRCCSpAJ 05] DA REPÚBLICA CASA CIVIL SUBCHEFIA PARA ASSUNTOS JURÍDICOS, P. **Medida Provisória 2.200-2**. Disponível em <http://www.planalto.gov.br/ccivil_03/MPV/Antigas_2001/2200–2.htm>. Acesso em: 14 de setembro de 2005.

[dSD 04] DA SILVA DIAS, J. **Confiança no Documento Eletrônico**. Florianópolis - SC: Universidade Federal de Santa Catarina, 2004. Tese de Doutorado.

[dTdI 05] DE TECNOLOGIA DA INFORMAÇÃO, I. N. **O que é assinatura digital**. Disponível em <http://www.iti.br>. Acesso em: 02 de abril de 2005.

[elo 05] **E–Lock ProSigner**. Disponível em <http://www.elock.com>. Acesso em: 10 de novembro de 2005.

[FRE 95] FREEDMAN, A. **Dicionário de Informática**. São Paulo: Makron Books, 1995.

[ITI 05] **Programas para Certificação Digital**. Disponível em <http://www.iti.br/twiki/bin/view/Main/Programas>. Acesso em: 26 de setembro de 2005.

[KAZ 03] KAZIENKO, J. F. **Assinatura Digital de Documentos Eletrônicos através da Impressão Digital**. Florianópolis - SC: Universidade Federal de Santa Catarina, 2003. Dissertação de Mestrado.

[LAR 00] LARMAN, C. **Utilizando UML e Padrões: uma introdução à análise e ao projeto orientados a objetos**. Porto Alegre: Bookman, 2000.

[LUC 86] LUCCHESI, C. L. **Introdução à Criptografia Computacional**. Campinas: Papirus Livraria Editora, 1986.

[MIG 02] MIGNONI, M. E. **Políticas e Declaração de Práticas de Certificação Digital para UFSC**. Florianópolis - SC: Universidade Federal de Santa Catarina, 2002. Dissertação de Mestrado.

[MOR 05] MORENO, E. D.; PEREIRA, F. D.; CHIARAMONTE, R. B. **Criptografia em Software e Hardware**. São Paulo: Novatec Editora, 2005.

[ntp 06] **JavaSntpClient - NTP**. Disponível em <http://ntp.isc.org/bin/view/Support/JavaSntpClient>. Acesso em: 22 de fevereiro de 2006.

[OAK 99] OAKS, S. **Segurança de Dados em Java**. Rio de Janeiro: Editora Ciência Moderna, 1999.

[ome 05] **Sistema Criptográfico Omega**. Disponível em <http://omega.trixhost.com/modules.php?name=News&file=article&sid=10>. Acesso em: 23 de novembro de 2005.

[PAS 02] PASQUAL, E. S. **IDDE - Uma Infra-estrutura para a Datação de Documentos Eletrônicos**. Florianópolis - SC: Universidade Federal de Santa Catarina, 2002. Dissertação de Mestrado.

[RIB 03] RIBEIRO, P. S. **Um Protocolo Criptográfico Para Comunicação Anônima Segura em Grupo**. Florianópolis - SC: Universidade Federal de Santa Catarina, 2003. Dissertação de Mestrado.

[SEB 03] SEBESTA, R. W. **Conceitos de Linguagens de Programação**. Porto Alegre: Bookman, 2003.

[THI 03] THING, L. **Dicionário de Tecnologia**. São Paulo: Editora Futura, 2003.

[VOL 01] VOLPI, M. M. **Assinatura Digital - Aspectos Técnicos, Práticos e Legais**. Rio de Janeiro: Axcel Books do Brasil Editora, 2001.

Apêndice A

Artigo

Software para Assinatura Digital

Evandro Araujo de Sousa

Departamento de Informática e Estatística
Universidade Federal de Santa Catarina - Florianópolis, SC - Brasil

evandro@inf.ufsc.br

Abstract. *This article presents a proposal of a software for digital signature, suggesting implementation in the programming language Java. To sign digitally an archive it is necessary to use algorithms of the assymmetrical cryptography. The Java possess a cryptography architecture, with classrooms and packages to implement and to use digital signature, cryptography keys and certificates. An alternative to the supplier of services of cryptography of the Sun is the supplier of the BouncyCastle, that implements digital signature in diverse format PKCS#7. Exists softwares of digital signature and the analysis of these softwares, it is the basic support for the proposal of a new software.*

Resumo. *Este artigo apresenta uma proposta de um software para assinatura digital, sugerindo a implementação na linguagem de programação Java. Para assinar digitalmente um arquivo é necessário utilizar algoritmos da criptografia*

assimétrica. O Java possui uma arquitetura de criptografia, com classes e pacotes para implementar e utilizar assinatura digital, chaves criptográficas e certificados. Uma alternativa ao provedor de serviços de criptografia da Sun é o provedor da BouncyCastle, que implementa assinatura digital no formato PKCS#7. Existem diversos softwares de assinatura digital e a análise destes softwares, é o apoio fundamental para a proposta de um novo software.

A.1 Introdução

Com o aumento do uso de correspondências eletrônicas, é evidente a substituição de documentos em papel pelos documentos eletrônicos, contudo é necessário garantir a segurança da tramitação destes documentos, isto pode ser obtido através da assinatura digital, resultando em uma correspondência segura, íntegra e autêntica para o destinatário.

O objetivo deste trabalho é a apresentação de uma proposta de um software para assinatura digital, através da implementação das funcionalidades presentes nas análises positivas e a implementação de novas funcionalidades que estejam relacionadas aos pontos negativos existentes nos softwares de assinatura digital.

A.2 Criptografia

A criptografia assimétrica é a utilizada na implementação da assinatura digital. É um sistema criptográfico que usa um par de chaves relacionadas, sendo uma chave pública, que é disponibilizada para qualquer pessoa e a outra é a chave privada, mantida em segredo.

O texto cifrado com a chave privada somente é decifrado com a chave pública, sendo possível a ocorrência do contrário, ou seja, o texto legível pode ser cifrado pela chave pública e o resultado, será decifrado com a chave privada.

A.3 Assinatura Digital

O objetivo da assinatura digital é substituir a assinatura manuscrita, oferecendo ao mundo digital as mesmas garantias do mundo real. A digitalização da assinatura manuscrita não é suficiente para alcançar esse propósito, pois a mesma pode ser copiada e anexada em qualquer documento.

A assinatura digital é usada para autenticar a identidade do remetente de uma mensagem, assegurando que o conteúdo original desta mensagem não será alterado.

Ela fica vinculada ao arquivo original, ocorrendo qualquer alteração a assinatura é inválida. Assim pode-se verificar a integridade do documento, permitindo ao destinatário perceber qualquer modificação feita no arquivo original.

A.3.1 Funcionamento

Para obter a assinatura digital de um arquivo é realizado a função resumo do arquivo que deseja assinar, após isso, cifra o resumo com a chave privada do remetente e finaliza enviando o arquivo, junto com o resultado da assinatura digital.

De modo inverso, para realizar a verificação de uma assinatura digital em um arquivo assinado deve-se primeiramente decifrar a assinatura digital com a chave pública do remetente, depois realiza a função resumo no arquivo original recebido e por último compara o resultado do deciframento da assinatura digital com o resultado da função resumo. A assinatura digital é válida se a comparação dos resultados forem idênticos.

A.3.2 Legalidade

Existe a necessidade da assinatura digital estar devidamente regulamentada, para que se tenha sua completa utilização, pois sem sua validade jurídica, o uso da assinatura digital fica restrita a poucos serviços.

Um documento assinado digitalmente é reconhecido da mesma forma que um documento assinado de próprio punho, conforme consta no § 1, do Art 10 da

Medida Provisória 2.200-2, de 24 de agosto de 2001: “As declarações constantes dos documentos em forma eletrônica produzidos com a utilização de processo de certificação disponibilizado pela ICP-Brasil presumem-se verdadeiros em relação aos signatários, na forma do Art 131 da Lei 3.071, de 01 de janeiro de 1916 - Código Civil”.

A.4 Java

A API de segurança faz parte do núcleo de API’s da linguagem de programação Java, fazendo parte do pacote java.security. Esta API é projetada para permitir a funcionalidade de baixo e alto nível de segurança no desenvolvimento de programas.

A arquitetura que forma a base da API de segurança em Java é baseada nas classes Provider e Security, que juntas formam um conjunto de mapeamentos que permite à API de segurança determinar dinamicamente o conjunto de classes que deve usar para implementar certas operações.

A classe Security centraliza todas as propriedades e métodos comuns de segurança. E uma de suas finalidades é o controle dos provedores de serviços de criptografia. Já a classe Provider representa um provedor para a API de segurança Java, ou seja, um provedor de serviços de criptografia que fornece algoritmos, geração de chaves e demais funcionalidades, permitindo implementações específicas.

A.4.1 Provedor BouncyCastle

O provedor da BouncyCastle é uma alternativa ao provedor da Sun, sendo livre e podendo ser utilizado comercialmente. Fornece implementações para os serviços de criptografia, contendo recursos para realizar uma assinatura digital no formato PKCS#7.

A BouncyCastle é um provedor que esta de acordo com a extensões de criptografia Java. A vantagem para programar alguma aplicação com este provedor, é que possui funcionalidades extras para a computação criptográfica e ainda pode ser desenvolvida em lugares que a exportação da criptografia é controlada.

A.5 Softwares de Assinatura Digital

A quantidade existente de utilitários para o uso de assinatura digital é grande, existindo softwares simples que utilizam apenas a linha de comando para executar suas tarefas e softwares com funcionalidades estendidas para outro tipo de software, como exemplo um editor de texto.

A análise de alguns softwares através de suas características, como a licença de uso, o sistema operacional exigido para ser instalado, o formato do arquivo resultado da assinatura, o gerenciamento de chaves e certificados e as demais funcionalidades apoiam a proposta de um novo software de assinatura digital.

Com o resultado desta análise foi possível verificar os pontos positivos e também os negativos dos softwares. Foi possível também verificar se os formatos dos arquivos resultados da assinatura podem ser interoperáveis.

Existem funcionalidades semelhantes em quase todos os softwares estudados. A diferença de um para o outro, é a facilidade apresentada para uma determinada atividade do usuário. Positivamente foi observado a facilidade de uso e o formato PKCS#7 do arquivo resultado da assinatura. Mas nem todos os softwares apresentam este formato, o que foi observado como um ponto negativo.

O ponto negativo de maior expressão é o custo para aquisição do software que exige licença de uso para o seu funcionamento. Ter como requisito de instalação apenas o sistema operacional Windows, também foi observado como um ponto negativo. A maioria dos softwares deixam a funcionalidade de gerenciamento de certificados e importação de chaves criptográficas a cargo do sistema operacional.

A.6 Proposta do Software

A proposta do software está definida na figura A.1. Seus propósitos são dispor funcionalidades que foram pontos positivos dos softwares analisados e apresentar uma implementação diferente para as funcionalidades observadas como pontos negativos.

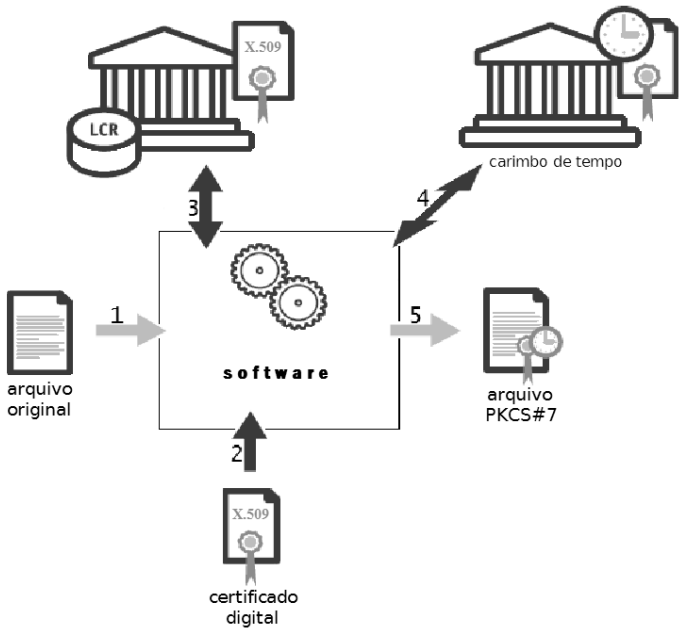

Figura A.1: Proposta do Software

Pela figura A.1 o usuário escolhe um arquivo para ser assinado. Após a escolha, o software lista os certificados do usuário e com a escolha de um certificado é feita uma consulta à Lista de Certificados Revogados para verificar a sua revogação. Não havendo restrições ao uso do certificado, o software requisita o carimbo de tempo a um servidor de data/hora. Finalizando apresenta o resultado da assinatura digital no formato PKCS#7.

Apêndice B

Código-Fonte

B.1 LEIAME

```
arquivos:

 - compilar.bat > compila as classes
 - compilarlinux.sh > versão linux

 - executar.bat > executa a classe Principal, a qual contém o método main
 - executarlinux.sh > versão linux

 - codigofonte > lista das classes para serem compiladas
 - codigofontelinux > versão linux

diretórios:

 - assina > diretórios das classes

 - keystores > diretório de armazenamento das keystores dos usuários

 - ON > contém o certificado do Observatório Nacional

 - Provider > contém o pacote .jar da BouncyCastle
```

B.2 compilar.bat

```
javac -classpath .;assina;assina\Arquivo;assina\Assinatura;assina\Carimbo;assina\Certific
ado;assina\Keystore;assina\SNTP;assina\Janela;assina\Usuario;Provider\bcprov-jdk15-131.ja
r @codigofonte
```

B.3 compilarlinux.sh

```
#!/bin/sh
javac -classpath .:assina/Arquivo:assina/Assinatura:assina/Carimbo:assina/Certificado:ass
ina/Keystore:assina/SNTP:assina/Janela:assina/Usuario:Provider/bcprov-jdk15-131.jar @codi
gofontelinux
```

B.4 executar.bat

```
java -classpath .;assina;assina\Janela;Provider\bcprov-jdk15-131.jar Principal
```

B.5 executarlinux.sh

```
#!/bin/sh
java -classpath .:assina:assina/Janela:Provider/bcprov-jdk15-131.jar Principal
```

B.6 codigofonte

```
assina\Arquivo\ArqByte.java assina\Arquivo\ArqObjeto.java assina\Assinatura\Assina.java a
ssina\Carimbo\CarimboTempo.java assina\Certificado\Certificado.java assina\Certificado\Cr
l.java assina\Janela\DialogoCriarUsuario.java assina\Janela\DialogoLogin.java assina\Arqu
ivo\Diretorio.java assina\Janela\JanelaPrincipal.java assina\Keystore\Keystore.java assin
a\SNTP\NtpMessage.java assina\Principal.java assina\SNTP\SntpClient.java assina\Assinatur
a\Verifica.java assina\Usuario\Usuario.java
```

B.7 codigofontelinux

```
assina/Arquivo/ArqByte.java assina/Arquivo/ArqObjeto.java assina/Assinatura/Assina.java a
ssina/Carimbo/CarimboTempo.java assina/Certificado/Certificado.java assina/Certificado/Cr
l.java assina/Janela/DialogoCriarUsuario.java assina/Janela/DialogoLogin.java assina/Arqu
ivo/Diretorio.java assina/Janela/JanelaPrincipal.java assina/Keystore/Keystore.java assin
a/SNTP/NtpMessage.java assina/Principal.java assina/SNTP/SntpClient.java assina/Assinatur
```

```
a/Verifica.java assina/Usuario/Usuario.java
```

B.8 ArqByte.java

```
1  package assina.Arquivo;
2
3  import java.io.*;
4  import java.io.FilterInputStream.*;
5  import java.util.*;
6  import java.security.*;
7
8  public class ArqByte {
9
10    public ArqByte() {
11    }
12
13    public byte[] entrarDados(String arq)
       throws ClassNotFoundException, FileNotFoundException, IOException {
14       FileInputStream fisArq = new FileInputStream(arq);
15       int x = fisArq.available();
16       byte[] b = new byte[x];
17       for (int i = 0 ; i < x; i++) {
18        b[i] = (byte) fisArq.read();
19       }
20       fisArq.close();
21       return b;
22    }
23
24    public void sairDados(byte[] b, String arq)
       throws ClassNotFoundException, FileNotFoundException, IOException {
25       FileOutputStream fosArq = new FileOutputStream(arq);
26       fosArq.write(b);
27       fosArq.close();
28    }
29 }
```

B.9 ArqObjeto.java

```
1  package assina.Arquivo;
2
3  import java.io.*;
4  import java.io.FilterInputStream.*;
```

```
5  import java.util.*;
6  import java.security.*;
7
8  public class ArqObjeto {
9
10    public ArqObjeto() {
11    }
12
13    public Timestamp entrarDadosCarimboTempo(String arq)
       throws ClassNotFoundException, FileNotFoundException, IOException {
14      Timestamp carimbo;
15      FileInputStream fisArq = new FileInputStream(arq);
16      ObjectInputStream oisArq = new ObjectInputStream(fisArq);
17      carimbo = (Timestamp) oisArq.readObject();
18      fisArq.close();
19      return carimbo;
20    }
21
22    public void sairDadosCarimboTempo(Timestamp carimbo, String arq)
       throws ClassNotFoundException, FileNotFoundException, IOException {
23      FileOutputStream fosArq = new FileOutputStream(arq);
24      ObjectOutputStream oosArq = new ObjectOutputStream(fosArq);
25      oosArq.writeObject(carimbo);
26      oosArq.flush();
27      fosArq.close();
28    }
29 }
```

B.10 Assina.java

```
1  package assina.Assinatura;
2
3  import java.io.*;
4  import java.util.*;
5  import java.security.*;
6  import java.security.cert.*;
7  import java.security.cert.Certificate;
8  import java.security.PrivateKey;
9  import org.bouncycastle.jce.PKCS7SignedData;
10 import org.bouncycastle.jce.provider.BouncyCastleProvider;
11 import assina.Arquivo.ArqByte;
12
13 public class Assina {
```

```
14
15    PKCS7SignedData pkcs7;
16
17    public Assina() {
18    }
19
20    public boolean assinar(PrivateKey chavePrivada,
       Certificate[] certificados, String arq, String data, String algo)
        throws SignatureException, InvalidKeyException, NoSuchProviderException,
         NoSuchAlgorithmException, ClassNotFoundException,
          FileNotFoundException, IOException {
21       byte[] arquivo, resultado;
22       ArqByte objArquivo  = new ArqByte();
23       arquivo = objArquivo.entrarDados(arq);
24       pkcs7 = new PKCS7SignedData(chavePrivada, certificados, algo);
25       pkcs7.update(arquivo, 0, arquivo.length);
26       arquivo = null;
27       arquivo = data.getBytes();
28       pkcs7.update(arquivo, 0, arquivo.length);
29       resultado = pkcs7.getEncoded();
30       objArquivo.sairDados(resultado, arq + ".pkcs7");
31       return true;
32    }
33 }
```

B.11 CarimboTempo.java

```
1  package assina.Carimbo;
2
3  import java.io.*;
4  import java.util.Date;
5  import java.security.Timestamp;
6  import java.security.cert.CertPath;
7  import org.bouncycastle.jce.provider.BouncyCastleProvider;
8  import assina.Arquivo.ArqByte;
9  import assina.Arquivo.ArqObjeto;
10 import assina.Certificado.Certificado;
11 import assina.SNTP.SntpClient;
12
13 public class CarimboTempo {
14
15    Timestamp carimbo;
16
```

```
17    public CarimboTempo() {
18    }
19
20    public CarimboTempo(Date data, CertPath certif) {
21       carimbo = new Timestamp(data, certif);
22    }
23
24    public String informa() {
25       return carimbo.toString();
26    }
27
28    public String informarData() {
29       return carimbo.getTimestamp().toString();
30    }
31
32    public void carimbar() {
33       Date data;
34       try {
35          SntpClient sntp = new SntpClient("200.20.186.75");
36          data = sntp.informaData();
37       } catch (IOException io) {
38          System.out.println("AVISO - sem acesso a internet");
39          data = new Date();
40          System.out.println("Utilizando a hora do computador\n");
41       }
42
43       Certificado cert = new Certificado();
44       CertPath certif = cert.informaCertPath(System.getProperty("user.dir")
          + System.getProperty("file.separator") + "ON"
           + System.getProperty("file.separator") + "on.p7b");
45       carimbo = new Timestamp(data, certif);
46    }
47
48    public void salvar(String arq) {
49       try {
50          ArqObjeto obj = new ArqObjeto();
51          obj.sairDadosCarimboTempo(carimbo, (arq + ".tstamp"));
52       } catch (Exception e) {
53          e.printStackTrace();
54       }
55    }
56
57    public Timestamp informarCarimboTempo(String arq) {
58       try {
59          ArqObjeto obj = new ArqObjeto();
```

```
60          return obj.entrarDadosCarimboTempo(arq);
61       } catch (Exception e) {
62          e.printStackTrace();
63          return null;
64       }
65    }
66 }
```

B.12 Certificado.java

```
1  package assina.Certificado;
2
3  import javax.security.auth.x500.X500Principal;
4  import java.security.cert.CertificateException;
5  import java.security.cert.CertificateFactory;
6  import java.security.cert.X509Certificate;
7  import java.security.cert.Certificate;
8  import java.security.cert.*;
9  import java.security.*;
10 import java.io.*;
11 import java.util.Date;
12 import java.security.cert.CertificateFactorySpi;
13 import org.bouncycastle.jce.provider.BouncyCastleProvider;
14 import org.bouncycastle.jce.provider.JDKX509CertificateFactory;
15 import assina.Keystore.Keystore;
16 import assina.SNTP.SntpClient;
17
18 public class Certificado {
19
20    private Certificate certificado;
21
22    public Certificado() {
23    }
24
25    public Certificado(File cert, String alias, Keystore keystore, char[] pass) {
26       try {
27          InputStream fisCert = new FileInputStream(cert.getPath());
28          BufferedInputStream bisCert = new BufferedInputStream(fisCert);
29          CertificateFactory certFab = CertificateFactory.getInstance("X.509");
30          while (bisCert.available() > 0) {
31             certificado = certFab.generateCertificate(bisCert);
32          }
33          bisCert.close();
```

```
34          keystore.recebaCertificado(alias, this.informaCertificado(), pass);
35       } catch (Exception e) {
36          e.printStackTrace();
37       }
38    }
39
40    public CertPath informaCertPath(String arquivo) {
41       try {
42          Security.addProvider(new BouncyCastleProvider());
43          InputStream is = new FileInputStream(arquivo);
44          JDKX509CertificateFactory certif = new JDKX509CertificateFactory();
45          CertPath cert = (CertPath)certif.engineGenerateCertPath(is, "PKCS7");
46          is.close();
47          return cert;
48       } catch (Exception e) {
49          e.printStackTrace();
50          return null;
51       }
52    }
53
54    public Certificate informaCertificado() {
55       return certificado;
56    }
57
58    public boolean informaValidade(Certificate certif) {
59       Date data;
60       try {
61          SntpClient sntp = new SntpClient("200.20.186.75");
62          data = sntp.informaData();
63       } catch (IOException io) {
64          System.out.println("AVISO - sem acesso a internet");
65          data = new Date();
66          System.out.println("Utilizando a hora do computador\n");
67       }
68       X509Certificate x509Certif = (X509Certificate)certif;
69       try {
70          x509Certif.checkValidity(data);
71          return true;
72       } catch (CertificateExpiredException cee) {
73          return false;
74       } catch (CertificateNotYetValidException cnyve) {
75          return false;
76       }
77    }
78 }
```

B.13 Crl.java

```
1  package assina.Certificado;
2
3  import java.io.*;
4  import java.net.*;
5  import java.net.URL;
6  import java.security.*;
7  import java.security.cert.*;
8  import java.security.cert.Certificate;
9  import java.security.cert.X509Certificate;
10 import org.bouncycastle.asn1.*;
11 import org.bouncycastle.asn1.x509.*;
12
13 public class Crl {
14
15    URL url;
16    X509CRL crl;
17    boolean temCRL;
18
19    public Crl(Certificate certif) {
20       try {
21          obterPontoDistrib((X509Certificate)certif);
22          if (temCRL) {
23             obterCRL();
24          }
25       } catch (Exception e) {
26          e.printStackTrace();
27       }
28    }
29
30    public void obterPontoDistrib(X509Certificate certificado)
       throws CertificateParsingException {
31       try {
32          DERObject obj = informaExtensao(certificado,
             X509Extensions.CRLDistributionPoints.getId());
33          if (obj == null) {
34             temCRL = false;
35          } else {
36             ASN1Sequence pontoDistrib = (ASN1Sequence) obj;
37             ASN1Sequence pDistrib = (ASN1Sequence) pontoDistrib.getObjectAt(0);
38             ASN1TaggedObject objTagged = (ASN1TaggedObject) pDistrib.getObjectAt(0);
39             String u = informaNomeGeral(objTagged.getObject());
40             String v = u.substring(0,5);
41             if (v.equals("https")) {
```

```
42                temCRL = false;
43             } else {
44                url= new URL(u);
45                temCRL = true;
46             }
47          }
48       } catch (Exception e) {
49          e.printStackTrace();
50       }
51    }
52
53    private DERObject informaExtensao(X509Certificate certif, String oid)
       throws IOException {
54       byte[] valor = certif.getExtensionValue(oid);
55       if (valor == null) {
56          return null;
57       }
58       ASN1InputStream ais = new ASN1InputStream(new ByteArrayInputStream(valor));
59       ASN1OctetString octetos = (ASN1OctetString) ais.readObject();
60       ais = new ASN1InputStream(new ByteArrayInputStream(octetos.getOctets()));
61       return ais.readObject();
62    }
63
64    private String informaNomeGeral(DERObject nome) {
65       ASN1Sequence nomes = ASN1Sequence.getInstance((ASN1TaggedObject)nome, false);
66       if (nomes.size() == 0) {
67          return null;
68       }
69       DERTaggedObject objTagged = (DERTaggedObject)nomes.getObjectAt(0);
70       return new String(ASN1OctetString.getInstance(objTagged, false).getOctets());
71    }
72
73    private void obterCRL() {
74       try {
75          InputStream is = url.openStream();
76          CertificateFactory certif = CertificateFactory.getInstance("X.509");
77          crl = (X509CRL)certif.generateCRL(is);
78       } catch (IOException io) {
79          System.out.println("AVISO - sem acesso a internet");
80          temCRL = false;
81       } catch (Exception e) {
82          e.printStackTrace();
83       }
84    }
85
```

```
86    public boolean informaRevogacao(Certificate certif) {
87       return crl.isRevoked(certif);
88    }
89
90    public boolean informaTemCRL() {
91       return temCRL;
92    }
93 }
```

B.14 DialogoCriarUsuario.java

```
1   package assina.Janela;
2
3   import javax.swing.*;
4   import java.awt.*;
5   import java.awt.event.*;
6   import java.security.*;
7   import java.io.*;
8   import assina.Arquivo.Diretorio;
9   import assina.Keystore.Keystore;
10  import assina.Usuario.Usuario;
11
12  public class DialogoCriarUsuario extends JDialog {
13
14     private final String SETE = "7";
15     private final String OITO = "8";
16
17     private Ouvinte ouvinte;
18     private JanelaPrincipal janelaPai;
19     private DialogoLogin dialogoPai;
20
21     private JPanel painelA;
22     private JPanel painelB;
23     private JTextField campoNome;
24     private JPasswordField campoSenha;
25     private JLabel rotuloNome, rotuloSenha;
26     private JButton botaoOk, botaoCanc;
27
28     public DialogoCriarUsuario(JDialog pai, String titulo, boolean modal,
        JanelaPrincipal jpai) {
29        super(pai, titulo, modal);
30        dialogoPai = (DialogoLogin)pai;
31        janelaPai = jpai;
```

```
32        iniciarComponentes();
33     }
34
35     private void fechaJanela() {
36        this.setVisible(false);
37     }
38
39     private void limparCampos() {
40        campoNome.setText("");
41        campoSenha.setText("");
42     }
43
44     private void setarTituloPai(String log) {
45        dialogoPai.setarTitulo(log);
46     }
47
48     protected boolean criarDiretorio(String dir) {
49        String caminho = System.getProperty("user.dir")
           + System.getProperty("file.separator") + "keystores";
50        Diretorio diretorio = new Diretorio();
51        return diretorio.criar(caminho
           + System.getProperty("file.separator") + dir);
52     }
53
54     protected void criar() {
55        String log = campoNome.getText();
56        char[] sen = campoSenha.getPassword();
57        try {
58           if (log.length() == 0 || Character.toString(sen[0]).length() == 0) {
59              return;
60           }
61        } catch (ArrayIndexOutOfBoundsException aioobe) {
62           return;
63        }
64        Keystore keystore = new Keystore(log);
65        Usuario usuario = new Usuario(log,sen);
66        if (keystore.existe()) {
67           JOptionPane.showMessageDialog(this, "usuário já existe",
              "erro", JOptionPane.ERROR_MESSAGE);
68        } else if (criarDiretorio(log)) {
69           try {
70              keystore.criaKeystore(usuario.informaSenha());
71           } catch (Exception e) {
72              e.printStackTrace();
73           }
```

```
74          fechaJanela();
75          dialogoPai.setVisible(false);
76          janelaPai.recebaObjs(keystore, usuario);
77          setarTituloPai(log);
78       }
79       limparCampos();
80    }
81
82    private void iniciarComponentes() {
83       ouvinte = new Ouvinte();
84       painelA = new JPanel();
85       painelB = new JPanel();
86       botaoOk = new JButton("Ok");
87       botaoCanc = new JButton("Cancelar");
88       rotuloNome = new JLabel("Fornecer nome de usuário da conta");
89       campoNome = new JTextField(20);
90       rotuloSenha = new JLabel("Fornecer senha do usuário");
91       campoSenha = new JPasswordField(20);
92       GridBagConstraints gridBagConstraints;
93       painelA.setLayout(new GridBagLayout());
94       gridBagConstraints = new GridBagConstraints();
95       gridBagConstraints.gridx = 0;
96       gridBagConstraints.gridy = 0;
97       gridBagConstraints.anchor = GridBagConstraints.WEST;
98       gridBagConstraints.insets = new Insets(20, 20, 0, 20);
99       painelA.add(rotuloNome, gridBagConstraints);
100      gridBagConstraints = new GridBagConstraints();
101      gridBagConstraints.gridx = 0;
102      gridBagConstraints.gridy = 1;
103      gridBagConstraints.insets = new Insets(3, 20, 20, 20);
104      campoNome.setMinimumSize(new Dimension(220, 19));
105      campoNome.setPreferredSize(new Dimension(220, 19));
106      painelA.add(campoNome, gridBagConstraints);
107      gridBagConstraints = new GridBagConstraints();
108      gridBagConstraints.gridx = 0;
109      gridBagConstraints.gridy = 2;
110      gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
111      gridBagConstraints.insets = new Insets(3, 20, 0, 20);
112      painelA.add(rotuloSenha, gridBagConstraints);
113      gridBagConstraints = new GridBagConstraints();
114      gridBagConstraints.gridx = 0;
115      gridBagConstraints.gridy = 3;
116      gridBagConstraints.insets = new Insets(3, 20, 20, 20);
117      campoSenha.setMinimumSize(new Dimension(220, 19));
118      campoSenha.setPreferredSize(new Dimension(220, 19));
```

```
119       painelA.add(campoSenha, gridBagConstraints);
120       painelB.setLayout(new FlowLayout(FlowLayout.CENTER, 15, 20));
121       botaoOk.setActionCommand(SETE);
122       botaoOk.addActionListener(ouvinte);
123       painelB.add(botaoOk);
124       botaoCanc.setActionCommand(OITO);
125       botaoCanc.addActionListener(ouvinte);
126       painelB.add(botaoCanc);
127       getContentPane().setLayout(new GridLayout(2, 0, 0, 0));
128       getContentPane().add(painelA);
129       getContentPane().add(painelB);
130       setDefaultCloseOperation(WindowConstants.HIDE_ON_CLOSE);
131       Dimension tamanhoTela = Toolkit.getDefaultToolkit().getScreenSize();
132       setBounds((tamanhoTela.width-400)/2, (tamanhoTela.height-300)/2, 400, 300);
133       setResizable(false);
134    }
135
136 /**
137  * Classe Interna Ouvinte
138  * conforme a ação do componente executa o método relacionado
139  **/
140
141 protected class Ouvinte implements ActionListener {
142    public void actionPerformed(ActionEvent evento) {
143       switch (Integer.parseInt(evento.getActionCommand())) {
144          case 7:
145             criar();
146             break;
147          case 8:
148            fechaJanela();
149      }
150    }
151 } // FIM Classe Ouvinte
152 }
```

B.15 DialogoLogin.java

```
1   package assina.Janela;
2
3   import javax.swing.*;
4   import javax.swing.border.*;
5   import java.awt.*;
6   import java.awt.event.*;
```

```
7   import java.security.*;
8   import java.io.*;
9   import java.util.*;
10  import assina.Arquivo.Diretorio;
11  import assina.Keystore.Keystore;
12  import assina.Usuario.Usuario;
13
14  public class DialogoLogin extends JDialog {
15
16     private final String QUATRO = "4";
17     private final String CINCO = "5";
18     private final String SEIS = "6";
19
20     private Ouvinte ouvinte;
21     private JanelaPrincipal janelaPai;
22     private DialogoCriarUsuario dialogoCriar;
23
24     private JPanel painel1, painel2;
25     private JButton botaoCriar, botaoOkLogin, botaoCancLogin;
26     private Choice campoLogin;
27     private JPasswordField campoSenhaLogin;
28     private JLabel rotuloLogin, rotuloSenhaLogin;
29
30     public DialogoLogin(JFrame pai, String tit, boolean modal) {
31        super(pai, tit, modal);
32        janelaPai = (JanelaPrincipal) pai;
33        iniciaComponentes();
34     }
35
36     protected void listaUsuarios() {
37        campoLogin.removeAll();
38        String caminho = System.getProperty("user.dir")
           + System.getProperty("file.separator") + "keystores";
39        Diretorio dir = new Diretorio();
40        String[] dirs = dir.listar(caminho);
41        for (int i = 0; i < dirs.length; i++) {
42           campoLogin.add(dirs[i]);
43        }
44     }
45
46     public void limparCampos() {
47        campoLogin.select(0);
48        campoSenhaLogin.setText("");
49     }
50
```

```
51     public void setarTitulo(String tit) {
52        janelaPai.setTitle("assinatura digital - " + tit);
53     }
54
55     public void mostrar() {
56        dialogoCriar = new DialogoCriarUsuario(this,
           "Criação de Usuário", true, janelaPai);
57     dialogoCriar.setVisible(true);
58     }
59
60     protected void logar() throws NoSuchAlgorithmException,
        ClassNotFoundException, FileNotFoundException, IOException {
61        String log = campoLogin.getSelectedItem();
62        char[] sen = campoSenhaLogin.getPassword();
63        try {
64           if (Character.toString(sen[0]).length() == 0) {
65              return;
66           }
67        } catch (ArrayIndexOutOfBoundsException aioobe) {
68           return;
69        }
70        Keystore keystore = new Keystore(log);
71        Usuario usuario = new Usuario(log,sen);
72        try {
73           if (!keystore.carregaKeystore(usuario.informaSenha())) {
74              JOptionPane.showMessageDialog(this, "senha incorreta",
                  "erro", JOptionPane.ERROR_MESSAGE);
75              limparCampos();
76           } else {
77              janelaPai.recebaObjs(keystore, usuario);
78              janelaPai.setTitle("assinatura digital - " + log);
79              setVisible(false);
80           }
81        } catch (Exception e) {
82           e.printStackTrace();
83        }
84     }
85
86     private void iniciaComponentes() {
87        ouvinte = new Ouvinte();
88        painel1 = new JPanel();
89        painel2 = new JPanel();
90        rotuloLogin = new JLabel("Login", SwingConstants.RIGHT);
91        campoLogin = new Choice();
92        listaUsuarios();
```

```
93        rotuloSenhaLogin = new JLabel("Senha", SwingConstants.RIGHT);
94        campoSenhaLogin = new JPasswordField(10);
95        painel1.setLayout(new GridLayout(2, 2, 10, 15));
96        painel1.add(rotuloLogin);
97        painel1.add(campoLogin);
98        painel1.add(rotuloSenhaLogin);
99        painel1.add(campoSenhaLogin);
100       botaoCriar = new JButton("Criar Usuário");
101       botaoCriar.setActionCommand(QUATRO);
102       botaoCriar.addActionListener(ouvinte);
103       botaoOkLogin = new JButton("Ok");
104       botaoOkLogin.setActionCommand(CINCO);
105       botaoOkLogin.addActionListener(ouvinte);
106       botaoCancLogin = new JButton("Cancelar");
107       botaoCancLogin.setActionCommand(SEIS);
108       botaoCancLogin.addActionListener(ouvinte);
109       painel2.setLayout(new FlowLayout(FlowLayout.CENTER));
110       painel2.add(botaoOkLogin);
111       painel2.add(botaoCancLogin);
112       painel2.add(botaoCriar);
113       getContentPane().setLayout(new BorderLayout(5,20));
114       getContentPane().add(painel1, BorderLayout.NORTH);
115       getContentPane().add(painel2, BorderLayout.CENTER);
116       Dimension tamanhoTela = Toolkit.getDefaultToolkit().getScreenSize();
117       setBounds((tamanhoTela.width-400)/2, (tamanhoTela.height-170)/2, 400, 170);
118       setResizable(false);
119       setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
120    }
121
122 /**
123  * Classe Interna Ouvinte
124  * executa um método conforme a ação do componente
125  **/
126
127 protected class Ouvinte implements ActionListener {
128    public void actionPerformed(ActionEvent evento) {
129       try {
130          switch (Integer.parseInt(evento.getActionCommand())) {
131             case 4:
132                mostrar();
133                break;
134             case 5:
135                logar();
136                break;
137             case 6:
```

```
138                System.exit(0);
139             }
140          } catch (NoSuchAlgorithmException nsae) {
141             System.err.println(nsae.toString());
142          } catch (ClassNotFoundException cnfe) {
143             System.err.println(cnfe.toString());
144          } catch (FileNotFoundException fnfe) {
145             System.err.println(fnfe.toString());
146          } catch (IOException io) {
147             System.err.println(io.toString());
148          }
149       }
150 } // FIM Classe Ouvinte
151 }
```

B.16 Diretorio.java

```
1  package assina.Arquivo;
2
3  import java.io.File;
4  import java.util.*;
5
6  public class Diretorio {
7
8     File diretorio;
9
10    public Diretorio() {
11    }
12
13    public String[] listar(String dir) {
14       return new File(dir).list();
15    }
16
17    public boolean criar(String dir) {
18       return new File(dir).mkdir();
19    }
20
21    public boolean salvarCertif(File arq, String dirKeysUsuario) {
22       String nomeArquivo = arq.getPath();
23       try {
24          if (nomeArquivo.endsWith(".p12") || nomeArquivo.endsWith(".pfx")) {
25             ArqByte arqByte = new ArqByte();
26             byte[] bite = arqByte.entrarDados(nomeArquivo);
```

```
27             arqByte.sairDados(bite, dirKeysUsuario +
                System.getProperty("file.separator") + arq.getName());
28             return true;
29          } else {
30             return false;
31          }
32       } catch (Exception e) {
33          e.printStackTrace();
34          return false;
35       }
36    }
37
38    public boolean removerCertif(String arq, String dirKeysUsuario) {
39       if (arq.endsWith(".p12") || arq.endsWith(".pfx")) {
40          File arquivo = new File(dirKeysUsuario +
             System.getProperty("file.separator") + arq);
41          arquivo.delete();
42          return true;
43       } else {
44          return false;
45       }
46    }
47 }
```

B.17 JanelaPrincipal.java

```
1   package assina.Janela;
2
3   import javax.swing.*;
4   import javax.swing.border.*;
5   import java.awt.*;
6   import java.awt.event.*;
7   import java.util.*;
8   import java.io.*;
9   import java.security.*;
10  import java.security.cert.*;
11  import java.security.cert.Certificate;
12  import assina.Arquivo.Diretorio;
13  import assina.Assinatura.Assina;
14  import assina.Assinatura.Verifica;
15  import assina.Carimbo.CarimboTempo;
16  import assina.Certificado.Certificado;
17  import assina.Certificado.Crl;
```

```
18 import assina.Keystore.Keystore;
19 import assina.Usuario.Usuario;
20
21 public class JanelaPrincipal extends JFrame {
22
23    private final String UM = "1";
24    private final String DOIS = "2";
25    private final String TRES = "3";
26    private final String NOVE = "9";
27    private final String ONZE = "11";
28    private final String DOZE = "12";
29    private final String TREZE = "13";
30    private final String CATORZE = "14";
31    private final String QUINZE = "15";
32    private final String DEZESSEIS = "16";
33    private final String DEZESSETE = "17";
34    private final String DEZOITO = "18";
35
36    private File arqAssinar;
37    private File arqVerificar;
38    private File arqCertif;
39
40    protected Ouvinte ouvinte;
41    protected Keystore keystore;
42    protected Usuario usuario;
43
44    private JToolBar barraBotoes;
45    private JTextField barraMensagens;
46    private JButton botaoAssinar;
47    private JButton botaoVerificar;
48    private JButton botaoCertif;
49    private JButton botaoSair;
50    private JPanel painel;
51    private PainelAssina painelAssinar;
52    private PainelVerifica painelVerificar;
53    private PainelCertif painelCertif;
54    private PainelFigura painelFigura;
55    private CardLayout leiaute;
56
57    private DialogoLogin login;
58
59    public JanelaPrincipal() {
60       inicia();
61       login = new DialogoLogin(this , "Validação de Acesso", true);
62       login.setVisible(true);
```

```
63        arqAssinar = new File("");
64        arqVerificar = new File("");
65        arqCertif = new File("");
66     }
67
68     public void recebaObjs(Keystore k, Usuario u) {
69        keystore = k;
70        usuario = u;
71     }
72
73     protected void mostrarPainelFigura() {
74        leiaute.show(painel, "painelFigura");
75     }
76
77     protected File caminho() {
78        File arq = new File(System.getProperty("user.dir"));
79        return arq;
80     }
81
82     public void inicia() {
83        ouvinte = new Ouvinte();
84        barraBotoes = new JToolBar();
85        botaoAssinar = new JButton("Assinar");
86        botaoVerificar = new JButton("Verificar");
87        botaoCertif = new JButton("Certificados");
88        botaoSair = new JButton("Sair");
89        barraMensagens = new JTextField();
90        painelAssinar = new PainelAssina();
91        painelVerificar = new PainelVerifica();
92        painelCertif = new PainelCertif();
93        painelFigura = new PainelFigura();
94        painel = new JPanel();
95        leiaute = new CardLayout();
96        getContentPane().setLayout(new java.awt.BorderLayout());
97        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
98        setTitle("software para assinatura digital");
99        setResizable(false);
100       Dimension tamanhoTela = Toolkit.getDefaultToolkit().getScreenSize();
101       setBounds((tamanhoTela.width-600)/2, (tamanhoTela.height-450)/2, 600, 450);
102       botaoAssinar.setActionCommand(UM);
103       botaoAssinar.addActionListener(ouvinte);
104       barraBotoes.add(botaoAssinar);
105       botaoVerificar.setActionCommand(DOIS);
106       botaoVerificar.addActionListener(ouvinte);
107       barraBotoes.add(botaoVerificar);
```

```
108       botaoCertif.setActionCommand(DEZESSEIS);
109       botaoCertif.addActionListener(ouvinte);
110       barraBotoes.add(botaoCertif);
111       botaoSair.setActionCommand(TRES);
112       botaoSair.addActionListener(ouvinte);
113       barraBotoes.add(botaoSair);
114       barraBotoes.setFloatable(false);
115       getContentPane().add(barraBotoes, BorderLayout.NORTH);
116       barraMensagens.setEditable(false);
117       barraMensagens.setBorder(new EtchedBorder());
118       getContentPane().add(barraMensagens, BorderLayout.SOUTH);
119       painel.setLayout(leiaute);
120       painel.add(painelFigura, "painelFigura");
121       painel.add(painelAssinar, "painelAssinar");
122       painel.add(painelVerificar, "painelVerificar");
123       painel.add(painelCertif, "painelCertif");
124       leiaute.first(painel);
125       getContentPane().add(painel, BorderLayout.CENTER);
126    }
127
128    /**
129     * Classe Interna PainelAssina
130     **/
131    public class PainelAssina extends JPanel {
132       private JButton botaoCancAss;
133       private JButton botaoEscolhaAss;
134       private JButton botaoOkAss;
135       private JTextField campoArquivoAss;
136       private JPanel painelChave;
137       private JPanel painelArquivoAss;
138       private JPanel painelOkCancAss;
139       private Choice campoChave;
140
141       public PainelAssina() {
142          iniciaAssina();
143       }
144
145       protected void limpaCampos() {
146          arqAssinar = new File("");
147          campoArquivoAss.setText("");
148       }
149
150       protected void escolherArq(String tit) {
151          JFileChooser dialogoArq = new JFileChooser();
152          dialogoArq.setDialogTitle(tit);
```

```
153          dialogoArq.setCurrentDirectory(caminho());
154          if (dialogoArq.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
155             arqAssinar = dialogoArq.getSelectedFile();
156             campoArquivoAss.setText(arqAssinar.getPath());
157          }
158       }
159
160       protected void mostrarChaves() {
161          campoChave.removeAll();
162          Vector vetor = new Vector();
163          try {
164             if (usuario.informaKeystores() == null) {
165                vetor = keystore.listaChave();
166             } else {
167                vetor = keystore.listaChaveDiversas(usuario.informaKeystores());
168             }
169          } catch (KeyStoreException kse) {
170             kse.printStackTrace();
171          }
172          for (int i = 0; i < vetor.size(); i++) {
173             campoChave.add((String)vetor.get(i));
174          }
175       }
176
177       private Keystore obterKeystore(String arq, char[] sen)
           throws IOException {
178          try {
179             arq = usuario.informaDirKeys() +
                  System.getProperty("file.separator") + arq;
180             return new Keystore(arq, sen, "PKCS12");
181          } catch (KeyStoreException kse) {
182             kse.printStackTrace();
183             return null;
184          } catch (NoSuchAlgorithmException nsae) {
185             nsae.printStackTrace();
186             return null;
187          } catch (CertificateException ce) {
188             ce.printStackTrace();
189             return null;
190          }
191       }
192
193       private void assinar() {
194          String algo;
195          PrivateKey priv;
```

```
196         Certificate[] certifs;
197         Certificate certif;
198         String arq = arqAssinar.getPath();
199         if (!arqAssinar.isFile()) {
200            return;
201         }
202         String alias = campoChave.getSelectedItem();
203         try {
204            if (alias.endsWith(".p12") || alias.endsWith(".pfx")) {
205               Keystore keystoreTemp;
206               char[] sen = null;
207               try {
208                  sen = (JOptionPane.showInputDialog(this,
                       "senha da keystore", "",
                        JOptionPane.QUESTION_MESSAGE)).toCharArray();
209                  if (Character.toString(sen[0]).length() == 0) {
210                     return;
211                  }
212               } catch (NullPointerException npe) {
213                  return;
214               } catch (ArrayIndexOutOfBoundsException aioobe) {
                     JOptionPane.showMessageDialog(this,
                      "digite uma senha", "erro", JOptionPane.ERROR_MESSAGE);
215                  return;
216               }
217               try {
218                  keystoreTemp = this.obterKeystore(alias, sen);
219               } catch (IOException ioe) {
220                  JOptionPane.showMessageDialog(this,
                       "senha errada", "erro", JOptionPane.ERROR_MESSAGE);
221                  return;
222               }
223               String aliasSel;
224               Vector vetor = keystoreTemp.listaChave();
225               Object[] aliases = vetor.toArray();
226               aliasSel = (String)
                   JOptionPane.showInputDialog(this,
                    "Escolha a chave privada", "",
                     JOptionPane.QUESTION_MESSAGE, null,
                      aliases, aliases[0]);
227               try {
228                  if (aliasSel.equals(null)) {
229                     return;
230                  }
231               } catch (NullPointerException npe) {
```

```
232                return;
233             }
234             priv = keystoreTemp.informaChavePriv(aliasSel,sen);
235             if (priv.getAlgorithm().equals("RSA")) {
236                algo = "MD5";
237             } else {
238                algo = "SHA";
239             }
240             try {
241                if (algo.equals(null)) {
242                   return;
243                }
244             } catch (NullPointerException npe) {
245                return;
246             }
247             certifs = keystoreTemp.informaCertificados(aliasSel);
248             certif = keystoreTemp.informaCertif(aliasSel);
249          } else {
250             char[] senha = null;
251             try {
252                senha = (JOptionPane.showInputDialog(this,
                    "senha da chave privada", "",
                     JOptionPane.QUESTION_MESSAGE)).toCharArray();
253                if (Character.toString(senha[0]).length() == 0) {
254                    return;
255                }
256             } catch (NullPointerException npe) {
257                return;
258             } catch (ArrayIndexOutOfBoundsException aioobe) {
259                JOptionPane.showMessageDialog(this,
                    "digite uma senha", "erro", JOptionPane.ERROR_MESSAGE);
260                return;
261             }
262             try {
263                priv = keystore.informaChavePriv(alias,senha);
264             } catch (UnrecoverableKeyException kse) {
265                JOptionPane.showMessageDialog(this,
                    "senha errada", "erro", JOptionPane.ERROR_MESSAGE);
266                return;
267             }
268             if (priv.getAlgorithm().equals("RSA")) {
269                algo = "MD5";
270             } else {
271                algo = "SHA";
272             }
```

```
273                try {
274                   if (algo.equals(null)) {
275                      return;
276                   }
277                } catch (NullPointerException npe) {
278                   return;
279                }
280                certifs = keystore.informaCertificados(alias);
281                certif = keystore.informaCertif(alias);
282             }
283             Certificado certificado = new Certificado();
284             if (!certificado.informaValidade(certif)) {
285                JOptionPane.showMessageDialog(this, "O certificado da "
                    + "chave privada é inválido",
                     "erro", JOptionPane.ERROR_MESSAGE);
286                this.limpaCampos();
287                return;
288             }
289             Crl crl = new Crl(certif);
290             if (!crl.informaTemCRL()) {
291                System.out.println("certificado não possui CRL"
                    + "\nOu CRL tem URL https");
292             } else {
293                if (crl.informaRevogacao(certif)) {
294                   JOptionPane.showMessageDialog(this,
                       "certificado revogado\n assinatura não realizada",
                        "erro", JOptionPane.ERROR_MESSAGE);
295                   this.limpaCampos();
296                   return;
297                }
298             }
299             CarimboTempo carimbo = new CarimboTempo();
300             carimbo.carimbar();
301             String data = carimbo.informarData();
302             Assina assina = new Assina();
303             if (assina.assinar(priv, certifs, arq, data, algo)) {
304                carimbo.salvar(arq);
305                JOptionPane.showMessageDialog(this,
                    ("resultado da assinatura\n"
                     + arqAssinar.getPath() + ".pkcs7"), null,
                      JOptionPane.INFORMATION_MESSAGE);
306             } else {
307                JOptionPane.showMessageDialog(this,
                    "assinatura não realizada", "erro",
                     JOptionPane.ERROR_MESSAGE);
```

```
308                this.limpaCampos();
309                return;
310             }
311             limpaCampos();
312          } catch (Exception e) {
313             e.printStackTrace();
314          }
315       }
316
317       private void iniciaAssina() {
318          Ouvinte ouvinte = new Ouvinte();
319          painelArquivoAss = new JPanel();
320          campoArquivoAss = new JTextField(30);
321          campoArquivoAss.setEditable(false);
322          botaoEscolhaAss = new JButton("Selecionar Arquivo ...");
323          botaoEscolhaAss.setActionCommand(NOVE);
324          botaoEscolhaAss.addActionListener(ouvinte);
325          painelChave = new JPanel();
326          campoChave = new Choice();
327          painelOkCancAss = new JPanel();
328          botaoOkAss = new JButton("Ok");
329          botaoOkAss.setActionCommand(ONZE);
330          botaoOkAss.addActionListener(ouvinte);
331          botaoCancAss = new JButton("Cancelar");
332          botaoCancAss.setActionCommand(DOZE);
333          botaoCancAss.addActionListener(ouvinte);
334          setLayout(new java.awt.GridLayout(3, 0));
335          painelArquivoAss.setBorder(new TitledBorder("Arquivo para Assinar"));
336          painelArquivoAss.add(campoArquivoAss);
337          painelArquivoAss.add(botaoEscolhaAss);
338          add(painelArquivoAss);
339          painelChave.setBorder(new TitledBorder("Chaves Privadas e Keystores do tipo PKCS12"));
340          painelChave.add(campoChave);
341          add(painelChave);
342          painelOkCancAss.add(botaoOkAss);
343          painelOkCancAss.add(botaoCancAss);
344          add(painelOkCancAss);
345       }
346    } // FIM Classe PainelAssina
347
348    /**
349     * Classe Interna PainelVerifica
350     **/
351    public class PainelVerifica extends JPanel {
352       private JButton botaoOkVerif;
```

```
353       private JButton botaoCancVerif;
354       private JButton botaoEscolhaVerif;
355       private JTextField campoArquivoVerif;
356       private JPanel painelArquivoVerif;
357       private JPanel painelOkCancVerif;
358
359       public PainelVerifica() {
360          iniciaVerifica();
361       }
362
363       protected void limpaCampos() {
364          arqVerificar = new File("");
365          campoArquivoVerif.setText("");
366       }
367
368       public void escolherArqVerif(String tit) {
369          JFileChooser dialogoArqVerif = new JFileChooser();
370          dialogoArqVerif.setDialogTitle(tit);
371          dialogoArqVerif.setCurrentDirectory(caminho());
372          if (dialogoArqVerif.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
373             arqVerificar = dialogoArqVerif.getSelectedFile();
374             JOptionPane.showMessageDialog(this,
                   "arquivo e assinatura devem estar no mesmo diretório",
                    null, JOptionPane.INFORMATION_MESSAGE);
375             campoArquivoVerif.setText(arqVerificar.getPath());
376          }
377       }
378
379       public void verificar() {
380          if (!arqVerificar.isFile()) {
381             return;
382          }
383          Verifica verifica = new Verifica();
384          try {
385             if (verifica.verificar(arqVerificar.getPath())) {
386                JOptionPane.showMessageDialog(this,
                      "verificação de assinatura correta",
                       null, JOptionPane.INFORMATION_MESSAGE);
387             } else {
388                JOptionPane.showMessageDialog(this,
                      "verificação de assinatura incorreta",
                       "erro", JOptionPane.ERROR_MESSAGE);
389             }
390          } catch (Exception e) {
391             e.printStackTrace();
```

```
392         }
393         this.limpaCampos();
394      }
395
396      private void iniciaVerifica() {
397         painelArquivoVerif = new JPanel();
398         botaoEscolhaVerif = new JButton("Selecionar Arquivo ...");
399         botaoEscolhaVerif.setActionCommand(TREZE);
400         botaoEscolhaVerif.addActionListener(ouvinte);
401         campoArquivoVerif = new JTextField(30);
402         campoArquivoVerif.setEditable(false);
403         painelOkCancVerif = new JPanel();
404         botaoOkVerif = new JButton("Ok");
405         botaoOkVerif.setActionCommand(CATORZE);
406         botaoOkVerif.addActionListener(ouvinte);
407         botaoCancVerif = new JButton("Cancelar");
408         botaoCancVerif.setActionCommand(QUINZE);
409         botaoCancVerif.addActionListener(ouvinte);
410         setLayout(new java.awt.GridLayout(2, 0));
411         painelArquivoVerif.setBorder(new TitledBorder("Arquivo para Verificar"));
412         painelArquivoVerif.add(campoArquivoVerif);
413         painelArquivoVerif.add(botaoEscolhaVerif);
414         add(painelArquivoVerif);
415         painelOkCancVerif.add(botaoOkVerif);
416         painelOkCancVerif.add(botaoCancVerif);
417         add(painelOkCancVerif);
418      }
419   } // FIM Classe PainelVerifica
420
421   /**
422    * Classe Interna PainelCertif
423    **/
424   public class PainelCertif extends JPanel {
425      private JToolBar barraOpcoes;
426      private JList lista;
427      private JButton botaoAdicionar;
428      private JButton botaoRemover;
429      private JTextField texto;
430
431      public PainelCertif() {
432         iniciaCertif();
433      }
434
435      public void informaLista() {
436         try {
```

```
437             if (usuario.informaKeystores() == null) {
438                lista.setListData(keystore.listaCertif());
439             } else {
440                lista.setListData(keystore.listaCertifDiversos(usuario.informaKeystores()));
441             }
442          } catch (Exception e) {
443             e.printStackTrace();
444          }
445       }
446
447       public void adicionaItemLista(String tit) {
448          String alias;
449          JFileChooser dialogoArqCertif = new JFileChooser();
450          dialogoArqCertif.setDialogTitle(tit);
451          dialogoArqCertif.setCurrentDirectory(caminho());
452          if (dialogoArqCertif.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
453             arqCertif = dialogoArqCertif.getSelectedFile();
454             Diretorio diretorio = new Diretorio();
455             if (diretorio.salvarCertif(arqCertif, usuario.informaDirKeys())) {
456                JOptionPane.showMessageDialog(this, "arquivo "
                    + arqCertif.getName() + " salvo", null,
                     JOptionPane.INFORMATION_MESSAGE);
457                this.informaLista();
458             } else {
459                alias = JOptionPane.showInputDialog(this,
                    "Digite um nome", "Alias do Certificado",
                     JOptionPane.PLAIN_MESSAGE);
460                try {
461                   Certificado certificado = new
462                   Certificado(arqCertif, alias, keystore,
463                   usuario.informaSenha());
464                   if (keystore.existeEntrada(alias)) {
465                      this.informaLista();
466                      JOptionPane.showMessageDialog(this,
                          "certificado " + alias + " adicionado",
                           null, JOptionPane.INFORMATION_MESSAGE);
467                   } else {
468                      JOptionPane.showMessageDialog(this,
                          "certificado não adicionado",
                           "erro", JOptionPane.ERROR_MESSAGE);
469                   }
470                } catch (Exception e) {
471                   e.printStackTrace();
472                }
473             }
```

```
474         }
475      }
476
477      public void removeItemLista() {
478         try {
479            int confirma;
480            String alias = (String)lista.getSelectedValue();
481            Diretorio diretorio = new Diretorio();
482            confirma = JOptionPane.showConfirmDialog(this, alias,
                "Remoção de Certificado", JOptionPane.WARNING_MESSAGE);
483            if (confirma == JOptionPane.OK_OPTION && (alias != null)) {
484               if (diretorio.removerCertif(alias,usuario.informaDirKeys())) {
485                  JOptionPane.showMessageDialog(this,"arquivo removido",
                      null, JOptionPane.INFORMATION_MESSAGE);
486               } else {
487                  keystore.deletaEntrada(alias, usuario.informaSenha());
488                  JOptionPane.showMessageDialog(this, "certificado removido",
                      null, JOptionPane.INFORMATION_MESSAGE);
489               }
490            } else if (confirma == JOptionPane.CANCEL_OPTION ||
                confirma == JOptionPane.CLOSED_OPTION) {
491               return;
492            } else {
493               JOptionPane.showMessageDialog(this, "selecionar um certificado"
                   + " na lista", null, JOptionPane.ERROR_MESSAGE);
494            }
495         } catch (Exception e){
496            e.printStackTrace();
497         }
498      }
499
500      private void iniciaCertif() {
501         lista = new JList();
502         barraOpcoes = new JToolBar();
503         botaoAdicionar = new JButton("Adicionar");
504         botaoAdicionar.setActionCommand(DEZESSETE);
505         botaoAdicionar.addActionListener(ouvinte);
506         botaoRemover = new JButton("Remover");
507         botaoRemover.setActionCommand(DEZOITO);
508         botaoRemover.addActionListener(ouvinte);
509         texto = new JTextField("\tLista de Chaves Privadas e Certificados");
510         setLayout(new BorderLayout());
511         add(lista, BorderLayout.CENTER);
512         barraOpcoes.add(botaoAdicionar);
513         barraOpcoes.add(texto);
```

```
514         barraOpcoes.add(botaoRemover);
515         barraOpcoes.setFloatable(false);
516         add(barraOpcoes, BorderLayout.NORTH);
517      }
518   } // FIM Classe PainelCertif
519
520   /**
521    * Classe Interna PainelFigura
522    **/
523   public class PainelFigura extends JPanel {
524      private JTextPane rotulo;
525
526      public PainelFigura() {
527         iniciaFigura();
528      }
529
530      private void iniciaFigura() {
531         rotulo = new JTextPane();
532         rotulo.setEditable(false);
533         rotulo.setBorder(new EtchedBorder());
534         rotulo.setText("\n\tUniversidade Federal de Santa Catarina\t\n\n"
               + "\tCentro Tecnológico\t\n\n"
                 + "\tDepartamento de Informática e Estatística\t\n\n"
                   + "\tCurso de Sistemas de Informação\n\n\n\n"
                     + "\tTrabalho de Conclusão de Curso - 2005/2\n\n\n\n"
                       + "\tAluno: Evandro Araujo\n\n"
                         + "\tOrientador: Prof. Ricardo Felipe Custódio\n");
535         add(rotulo);
536      }
537   } // FIM Classe PainelFigura
538
539   /**
540    * Classe Interna Ouvinte
541    **/
542
543   protected class Ouvinte implements ActionListener {
544
545      public void actionPerformed(ActionEvent evento) {
546         switch (Integer.parseInt(evento.getActionCommand())) {
547         case 1:
548                painelAssinar.mostrarChaves();
549                painelVerificar.limpaCampos();
550                leiaute.show(painel,"painelAssinar");
551                break;
552         case 2:
```

```
553                painelAssinar.limpaCampos();
554                leiaute.show(painel,"painelVerificar");
555                break;
556         case 3:
557                System.exit(0);
558                break;
559         case 9:
560                painelAssinar.escolherArq("Escolher Arquivo para Assinatura");
561                break;
562         case 11:
563                 painelAssinar.assinar();
564                 break;
565         case 12:
566                 painelAssinar.limpaCampos();
567                 mostrarPainelFigura();
568                 break;
569         case 13:
570                 painelVerificar.escolherArqVerif("Escolher Arquivo para Verificação");
571                 break;
572         case 14:
573                 painelVerificar.verificar();
574                 break;
575         case 15:
576                 painelVerificar.limpaCampos();
577                 mostrarPainelFigura();
578                 break;
579         case 16:
580                 painelAssinar.limpaCampos();
581                 painelVerificar.limpaCampos();
582                 leiaute.show(painel,"painelCertif");
583                 painelCertif.informaLista();
584                 break;
585         case 17:
586                 painelCertif.adicionaItemLista("Escolher Certificado");
587                 break;
588         case 18:
589                 painelCertif.removeItemLista();
590                 painelCertif.informaLista();
591         }
592       }
593    } // FIM Classe Ouvinte
594 }
```

B.18 Keystore.java

```
1   package assina.Keystore;
2
3   import java.security.KeyStore.*;
4   import java.security.cert.Certificate;
5   import java.security.cert.*;
6   import java.security.*;
7   import java.security.Provider;
8   import java.io.*;
9   import java.io.File;
10  import java.util.*;
11  import org.bouncycastle.jce.provider.BouncyCastleProvider;
12
13  public class Keystore {
14
15     private KeyStore keystore;
16     private File arqKeystore;
17
18     public Keystore(String ks) {
19        try {
20           Security.addProvider(new BouncyCastleProvider());
21           String caminho = System.getProperty("user.dir")
              + System.getProperty("file.separator") + "keystores"
               + System.getProperty("file.separator") + ks
                + System.getProperty("file.separator") + "keystore.bks";
22           arqKeystore = new File(caminho);
23           keystore = KeyStore.getInstance("bks");
24        } catch (KeyStoreException kse) {
25           System.err.println(kse.toString());
26        }
27     }
28
29     public Keystore(String arq, char[] pass, String tipo) throws
        KeyStoreException, IOException, NoSuchAlgorithmException,
         CertificateException {
30        keystore = KeyStore.getInstance(tipo);
31        FileInputStream fisKS = new FileInputStream(new File(arq));
32        keystore.load(fisKS, pass);
33     }
34
35     public boolean carregaKeystore(char[] pass) throws IOException,
        NoSuchAlgorithmException, CertificateException {
36        try {
37           FileInputStream isKS = new FileInputStream(arqKeystore);
```

```
38          keystore.load(isKS,pass);
39          isKS.close();
40          return true;
41       } catch (Exception e) {
42          e.printStackTrace();
43          return false;
44       }
45    }
46
47    public void criaKeystore(char[] pass) throws IOException, KeyStoreException,
       NoSuchAlgorithmException, CertificateException {
48       keystore.load(null,null);
49       OutputStream osKS = new FileOutputStream(arqKeystore);
50       keystore.store(osKS,pass);
51    }
52
53    public void recebaCertificado(String alias, Certificate cert, char[] pass)
       throws IOException, NoSuchAlgorithmException, CertificateException,
        KeyStoreException {
54       try {
55          keystore.setCertificateEntry(alias, cert);
56          OutputStream osKS = new FileOutputStream(arqKeystore);
57          keystore.store(osKS,pass);
58       } catch (Exception e) {
59          System.err.println(e.toString());
60       }
61    }
62
63    public boolean existe() {
64       return arqKeystore.exists();
65    }
66
67    public Vector listaChave() throws KeyStoreException {
68       Enumeration lista = keystore.aliases();
69       Vector vetor = new Vector();
70       while(lista.hasMoreElements()) {
71          String alias = (String) lista.nextElement();
72          if (keystore.isKeyEntry(alias)) {
73             vetor.addElement(alias);
74          }
75       }
76       return vetor;
77    }
78
79    public Vector listaChaveDiversas(String[] keystores)
```

```
        throws KeyStoreException {
80        Enumeration lista = keystore.aliases();
81        Vector vetor = new Vector();
82        try {
83           while(lista.hasMoreElements()) {
84              String alias = (String) lista.nextElement();
85              if (keystore.isKeyEntry(alias)) {
86                 vetor.addElement(alias);
87              }
88           }
89           int i = 0;
90           while (i < keystores.length) {
91              vetor.addElement(keystores[i]);
92              i++;
93           }
94           return vetor;
95        } catch (Exception e) {
96           System.err.println(e.toString());
97           throw new KeyStoreException();
98        }
99     }
100
101    public Vector listaCertif() throws KeyStoreException {
102       Enumeration lista = keystore.aliases();
103       Vector vetor = new Vector();
104       while(lista.hasMoreElements()) {
105          String alias = (String) lista.nextElement();
106          vetor.addElement(alias);
107       }
108       int i = 0;
109       return vetor;
110    }
111
112    public Vector listaCertifDiversos(String[] keystores) {
113       try {
114          Enumeration lista = keystore.aliases();
115          Vector vetor = new Vector();
116          while(lista.hasMoreElements()) {
117             String alias = (String) lista.nextElement();
118             vetor.addElement(alias);
119          }
120          int i = 0;
121          while (i < keystores.length) {
122             vetor.addElement(keystores[i]);
123             i++;
```

```
124          }
125          return vetor;
126       } catch (KeyStoreException kse) {
127          System.err.println(kse.toString());
128          return null;
129       }
130    }
131
132    public void deletaEntrada(String alias, char[] pass)
        throws KeyStoreException, FileNotFoundException, IOException,
         NoSuchAlgorithmException, CertificateException {
133       keystore.deleteEntry(alias);
134       OutputStream osKS = new FileOutputStream(arqKeystore);
135       keystore.store(osKS,pass);
136    }
137
138    public Certificate informaCertif(String alias) throws KeyStoreException {
139       return keystore.getCertificate(alias);
140    }
141
142    public PrivateKey informaChavePriv(String alias, char[] sen)
        throws KeyStoreException, NoSuchAlgorithmException,
         UnrecoverableKeyException  {
143       return (PrivateKey)keystore.getKey(alias, sen);
144    }
145
146    public Certificate[] informaCertificados(String alias)
        throws KeyStoreException {
147       return (Certificate[])keystore.getCertificateChain(alias);
148    }
149
150    public boolean existeEntrada(String alias) throws KeyStoreException {
151       return keystore.containsAlias(alias);
152    }
153 }
```

B.19 NtpMessage.java

```
1   /**
2    * modificação
3    *
4    *****************/
5   package assina.SNTP;
```

```
6
7   import java.text.DecimalFormat;
8   import java.text.SimpleDateFormat;
9   import java.util.Date;
10
11  /**
12   * This class represents a NTP message, as specified in RFC 2030.  The message
13   * format is compatible with all versions of NTP and SNTP.
15   *
16   * This class does not support the optional authentication protocol, and
17   * ignores the key ID and message digest fields.
18   *
19   * For convenience, this class exposes message values as native Java types, not
20   * the NTP-specified data formats.  For example, timestamps are
21   * stored as doubles (as opposed to the NTP unsigned 64-bit fixed point
22   * format).
23   *
24   * However, the contructor NtpMessage(byte[]) and the method toByteArray()
25   * allow the import and export of the raw NTP message format.
26   *
27   *
28   * Usage example
29   *
30   * // Send message
31   * DatagramSocket socket = new DatagramSocket();
32   * InetAddress address = InetAddress.getByName("ntp.cais.rnp.br");
33   * byte[] buf = new NtpMessage().toByteArray();
34   * DatagramPacket packet = new DatagramPacket(buf, buf.length, address, 123);
35   * socket.send(packet);
35   *
36   * // Get response
37   * socket.receive(packet);
38   * System.out.println(msg.toString());
39   *
40   *
41   * This code is copyright (c) Adam Buckley 2004
42   *
43   * This program is free software; you can redistribute it and/or modify it
44   * under the terms of the GNU General Public License as published by the Free
45   * Software Foundation; either version 2 of the License, or (at your option)
46   * any later version.  A HTML version of the GNU General Public License can be
47   * seen at http://www.gnu.org/licenses/gpl.html
48   *
49   * This program is distributed in the hope that it will be useful, but WITHOUT
50   * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
```

```
51   * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for
52   * more details.
53   *
54   *
55   * Comments for member variables are taken from RFC2030 by David Mills,
56   * University of Delaware.
57   *
58   * Number format conversion code in NtpMessage(byte[] array) and toByteArray()
59   * inspired by http://www.pps.jussieu.fr/~jch/enseignement/reseaux/
60   * NTPMessage.java which is copyright (c) 2003 by Juliusz Chroboczek
61   *
62   * @author Adam Buckley
63   */
64  public class NtpMessage
65  {
66  /**
67   * This is a two-bit code warning of an impending leap second to be
68   * inserted/deleted in the last minute of the current day.  It's values
69   * may be as follows:
70   *
71   * Value     Meaning
72   * -----     -------
73   * 0         no warning
74   * 1         last minute has 61 seconds
75   * 2         last minute has 59 seconds)
76   * 3         alarm condition (clock not synchronized)
77   */
78  public byte leapIndicator = 0;
79
80
81  /**
82   * This value indicates the NTP/SNTP version number.  The version number
83   * is 3 for Version 3 (IPv4 only) and 4 for Version 4 (IPv4, IPv6 and OSI).
84   * If necessary to distinguish between IPv4, IPv6 and OSI, the
85   * encapsulating context must be inspected.
86   */
87  public byte version = 3;
88
89
90  /**
91   * This value indicates the mode, with values defined as follows:
92   *
93   * Mode     Meaning
94   * ----     -------
95   * 0        reserved
```

```
96   * 1        symmetric active
97   * 2        symmetric passive
98   * 3        client
99   * 4        server
100  * 5        broadcast
101  * 6        reserved for NTP control message
102  * 7        reserved for private use
103  *
104  * In unicast and anycast modes, the client sets this field to 3 (client)
105  * in the request and the server sets it to 4 (server) in the reply. In
106  * multicast mode, the server sets this field to 5 (broadcast).
107  */
108  public byte mode = 0;
109
110
111 /**
112  * This value indicates the stratum level of the local clock, with values
113  * defined as follows:
114  *
115  * Stratum  Meaning
116  * ----------------------------------------------
117  * 0        unspecified or unavailable
118  * 1        primary reference (e.g., radio clock)
119  * 2-15     secondary reference (via NTP or SNTP)
120  * 16-255   reserved
121  */
122 public short stratum = 0;
123
124
125 /**
126  * This value indicates the maximum interval between successive messages,
127  * in seconds to the nearest power of two. The values that can appear in
128  * this field presently range from 4 (16 s) to 14 (16284 s); however, most
129  * applications use only the sub-range 6 (64 s) to 10 (1024 s).
130  */
131 public byte pollInterval = 0;
132
133
134 /**
135  * This value indicates the precision of the local clock, in seconds to
136  * the nearest power of two.  The values that normally appear in this field
137  * range from -6 for mains-frequency clocks to -20 for microsecond clocks
138  * found in some workstations.
139  */
140 public byte precision = 0;
```

```
141
142
143 /**
144  * This value indicates the total roundtrip delay to the primary reference
145  * source, in seconds.  Note that this variable can take on both positive
146  * and negative values, depending on the relative time and frequency
147  * offsets. The values that normally appear in this field range from
148  * negative values of a few milliseconds to positive values of several
149  * hundred milliseconds.
150  */
151 public double rootDelay = 0;
152
153
154 /**
155  * This value indicates the nominal error relative to the primary reference
156  * source, in seconds.  The values  that normally appear in this field
157  * range from 0 to several hundred milliseconds.
158  */
159 public double rootDispersion = 0;
160
161
162 /**
162  * This is a 4-byte array identifying the particular reference source.
163  * In the case of NTP Version 3 or Version 4 stratum-0 (unspecified) or
164  * stratum-1 (primary) servers, this is a four-character ASCII string, left
165  * justified and zero padded to 32 bits. In NTP Version 3 secondary
166  * servers, this is the 32-bit IPv4 address of the reference source. In NTP
167  * Version 4 secondary servers, this is the low order 32 bits of the latest
168  * transmit timestamp of the reference source. NTP primary (stratum 1)
169  * servers should set this field to a code identifying the external
170  * reference source according to the following list. If the external
171  * reference is one of those listed, the associated code should be used.
172  * Codes for sources not listed can be contrived as appropriate.
173  *
174  * Code     External Reference Source
175  * ----     -------------------------
176  * LOCL     uncalibrated local clock used as a primary reference for
177  *          a subnet without external means of synchronization
178  * PPS      atomic clock or other pulse-per-second source
179  *          individually calibrated to national standards
180  * ACTS     NIST dialup modem service
181  * USNO     USNO modem service
182  * PTB      PTB (Germany) modem service
183  * TDF      Allouis (France) Radio 164 kHz
184  * DCF      Mainflingen (Germany) Radio 77.5 kHz
```

```
185  * MSF      Rugby (UK) Radio 60 kHz
186  * WWV      Ft. Collins (US) Radio 2.5, 5, 10, 15, 20 MHz
187  * WWVB     Boulder (US) Radio 60 kHz
188  * WWVH     Kaui Hawaii (US) Radio 2.5, 5, 10, 15 MHz
189  * CHU      Ottawa (Canada) Radio 3330, 7335, 14670 kHz
190  * LORC     LORAN-C radionavigation system
191  * OMEG     OMEGA radionavigation system
192  * GPS      Global Positioning Service
193  * GOES     Geostationary Orbit Environment Satellite
194  */
195 public byte[] referenceIdentifier = {0, 0, 0, 0};
196
197
198 /**
199  * This is the time at which the local clock was last set or corrected, in
200  * seconds since 00:00 1-Jan-1900.
201  */
202 public double referenceTimestamp = 0;
203
204
205 /**
206  * This is the time at which the request departed the client for the
207  * server, in seconds since 00:00 1-Jan-1900.
208  */
209 public double originateTimestamp = 0;
210
211
212 /**
213  * This is the time at which the request arrived at the server, in seconds
214  * since 00:00 1-Jan-1900.
215  */
216 public double receiveTimestamp = 0;
217
218
219 /**
220  * This is the time at which the reply departed the server for the client,
221  * in seconds since 00:00 1-Jan-1900.
222  */
223 public double transmitTimestamp = 0;
224
225
226
227 /**
228  * Constructs a new NtpMessage from an array of bytes.
229  */
```

```
230 public NtpMessage(byte[] array)
231 {
232    // See the packet format diagram in RFC 2030 for details
233    leapIndicator = (byte) ((array[0] >> 6) & 0x3);
234    version = (byte) ((array[0] >> 3) & 0x7);
235    mode = (byte) (array[0] & 0x7);
236    stratum = unsignedByteToShort(array[1]);
237    pollInterval = array[2];
238    precision = array[3];
239
240    rootDelay = (array[4] * 256.0) +
        unsignedByteToShort(array[5]) +
         (unsignedByteToShort(array[6]) / 256.0) +
          (unsignedByteToShort(array[7]) / 65536.0);
241
242    rootDispersion = (unsignedByteToShort(array[8]) * 256.0) +
        unsignedByteToShort(array[9]) +
        (unsignedByteToShort(array[10]) / 256.0) +
         (unsignedByteToShort(array[11]) / 65536.0);
243
244    referenceIdentifier[0] = array[12];
245    referenceIdentifier[1] = array[13];
246    referenceIdentifier[2] = array[14];
247    referenceIdentifier[3] = array[15];
248
249    referenceTimestamp = decodeTimestamp(array, 16);
250    originateTimestamp = decodeTimestamp(array, 24);
251    receiveTimestamp = decodeTimestamp(array, 32);
252    transmitTimestamp = decodeTimestamp(array, 40);
253 }
254
255
256
257 /**
258  * Constructs a new NtpMessage in client -> server mode, and sets the
259  * transmit timestamp to the current time.
260  */
261 public NtpMessage()
262 {
263    // Note that all the other member variables are already set with
264    // appropriate default values.
265    this.mode = 3;
266    this.transmitTimestamp = (System.currentTimeMillis()/1000.0) + 2208988800.0;
267 }
268
```

```
269
270
271 /**
272  * This method constructs the data bytes of a raw NTP packet.
273  */
274 public byte[] toByteArray()
275 {
276    // All bytes are automatically set to 0
277    byte[] p = new byte[48];
278
279    p[0] = (byte) (leapIndicator << 6 | version << 3 | mode);
280    p[1] = (byte) stratum;
281    p[2] = (byte) pollInterval;
282    p[3] = (byte) precision;
283
284    // root delay is a signed 16.16-bit FP, in Java an int is 32-bits
285    int l = (int) (rootDelay * 65536.0);
286    p[4] = (byte) ((l >> 24) & 0xFF);
287    p[5] = (byte) ((l >> 16) & 0xFF);
288    p[6] = (byte) ((l >> 8) & 0xFF);
289    p[7] = (byte) (l & 0xFF);
290
291    // root dispersion is an unsigned 16.16-bit FP, in Java there are no
292    // unsigned primitive types, so we use a long which is 64-bits
293    long ul = (long) (rootDispersion * 65536.0);
294    p[8] = (byte) ((ul >> 24) & 0xFF);
295    p[9] = (byte) ((ul >> 16) & 0xFF);
296    p[10] = (byte) ((ul >> 8) & 0xFF);
297    p[11] = (byte) (ul & 0xFF);
298
299    p[12] = referenceIdentifier[0];
300    p[13] = referenceIdentifier[1];
301    p[14] = referenceIdentifier[2];
302    p[15] = referenceIdentifier[3];
303
304    encodeTimestamp(p, 16, referenceTimestamp);
305    encodeTimestamp(p, 24, originateTimestamp);
306    encodeTimestamp(p, 32, receiveTimestamp);
307    encodeTimestamp(p, 40, transmitTimestamp);
308
309    return p;
310 }
311
312
313
```

```
314 /**
315  * Returns a string representation of a NtpMessage
316  */
317 public String toString()
318 {
319    String precisionStr =
        new DecimalFormat("0.#E0").format(Math.pow(2, precision));
320
321 return "Leap indicator: " + leapIndicator + "\n" +
   "Version: " + version + "\n" +
   "Mode: " + mode + "\n" +
   "Stratum: " + stratum + "\n" +
   "Poll: " + pollInterval + "\n" +
   "Precision: " + precision + " (" + precisionStr + " seconds)\n" +
   "Root delay: " + new DecimalFormat("0.00").format(rootDelay*1000) + " ms\n" +
   "Root dispersion: " + new DecimalFormat("0.00").format(rootDispersion*1000) + " ms\n" +
   "Reference identifier: " + referenceIdentifierToString(referenceIdentifier, stratum, version) + "\n" +
   "Reference timestamp: " + timestampToString(referenceTimestamp) + "\n" +
   "Originate timestamp: " + timestampToString(originateTimestamp) + "\n" +
   "Receive timestamp:   " + timestampToString(receiveTimestamp) + "\n" +
   "Transmit timestamp:  " + timestampToString(transmitTimestamp);
322 }
323
324
325
326 /**
327  * Converts an unsigned byte to a short.  By default, Java assumes that
328  * a byte is signed.
329  */
330 public static short unsignedByteToShort(byte b)
331 {
332    if((b & 0x80)==0x80) return (short) (128 + (b & 0x7f));
333    else return (short) b;
334 }
335
336
337
338 /**
339  * Will read 8 bytes of a message beginning at <code>pointer</code>
340  * and return it as a double, according to the NTP 64-bit timestamp
341  * format.
342  */
343 public static double decodeTimestamp(byte[] array, int pointer)
344 {
345    double r = 0.0;
```

```
346
347    for(int i=0; i<8; i++)
348    {
349       r += unsignedByteToShort(array[pointer+i]) * Math.pow(2, (3-i)*8);
350    }
351
352    return r;
353 }
354
355
356
357 /**
358  * Encodes a timestamp in the specified position in the message
359  */
360 public static void encodeTimestamp(byte[] array, int pointer, double timestamp)
361 {
362    // Converts a double into a 64-bit fixed point
363    for(int i=0; i<8; i++)
364    {
365       // 2^24, 2^16, 2^8, .. 2^-32
366       double base = Math.pow(2, (3-i)*8);
367
368       // Capture byte value
369       array[pointer+i] = (byte) (timestamp / base);
370
371       // Subtract captured value from remaining total
372       timestamp = timestamp - (double) (unsignedByteToShort(array[pointer+i]) * base);
373    }
374
375    // From RFC 2030: It is advisable to fill the non-significant
376    // low order bits of the timestamp with a random, unbiased
377    // bitstring, both to avoid systematic roundoff errors and as
378    // a means of loop detection and replay detection.
379    array[7] = (byte) (Math.random()*255.0);
380 }
381
382
383
384 /**
385  * Returns a timestamp (number of seconds since 00:00 1-Jan-1900) as a
386  * formatted date/time string.
387  */
388 public static String timestampToString(double timestamp)
389 {
390    if(timestamp==0) return "0";
```

```
391
392    // timestamp is relative to 1900, utc is used by Java and is relative
393    // to 1970
394    double utc = timestamp - (2208988800.0);
395
396    // milliseconds
397    long ms = (long) (utc * 1000.0);
398
399    // date/time
400    String date = new SimpleDateFormat("dd-MMM-yyyy HH:mm:ss").format(new Date(ms));
401
402    // fraction
403    double fraction = timestamp - ((long) timestamp);
404    String fractionSting = new DecimalFormat(".000000").format(fraction);
405
406    return date + fractionSting;
407 }
408
409 /**
410  * modificação
411  *
412  * método incluso para retornar a data
413  *
414  ************************************/
415 public Date data(double timestamp) {
416    double utc = timestamp - (2208988800.0);
417    long ms = (long) (utc * 1000.0);
418    Date data = new Date(ms);
419    return data;
420 }
421
422
423
424 /**
425  * Returns a string representation of a reference identifier according
426  * to the rules set out in RFC 2030.
427  */
428 public static String referenceIdentifierToString(byte[] ref, short stratum, byte version)
429 {
430    // From the RFC 2030:
431    // In the case of NTP Version 3 or Version 4 stratum-0 (unspecified)
432    // or stratum-1 (primary) servers, this is a four-character ASCII
433    // string, left justified and zero padded to 32 bits.
434    if(stratum==0 || stratum==1)
435    {
```

```
436       return new String(ref);
437    }
438
439    // In NTP Version 3 secondary servers, this is the 32-bit IPv4
440    // address of the reference source.
441    else if(version==3)
442    {
443       return unsignedByteToShort(ref[0]) + "." +
           unsignedByteToShort(ref[1]) + "." +
            unsignedByteToShort(ref[2]) + "." +
             unsignedByteToShort(ref[3]);
444    }
445
446    // In NTP Version 4 secondary servers, this is the low order 32 bits
447    // of the latest transmit timestamp of the reference source.
448    else if(version==4)
449    {
450    return "" + ((unsignedByteToShort(ref[0]) / 256.0) +
        (unsignedByteToShort(ref[1]) / 65536.0) +
         (unsignedByteToShort(ref[2]) / 16777216.0) +
          (unsignedByteToShort(ref[3]) / 4294967296.0));
451    }
452
453    return "";
454 }
455 }
```

B.20 Principal.java

```
1 import assina.Janela.JanelaPrincipal;
2
3 public class Principal {
4
5    public static void main(String a[]) {
6       JanelaPrincipal janela = new JanelaPrincipal();
7       janela.setVisible(true);
8    }
9 }
```

B.21 SntpClient.java

```
1   /**
2    * modificação
3    *
4    ******************/
5   package assina.SNTP;
6
7   import java.io.IOException;
8   import java.net.DatagramPacket;
9   import java.net.DatagramSocket;
10  import java.net.InetAddress;
11  import java.text.DecimalFormat;
12  /**
13   * modificação
14   *************/
15  import java.util.Date;
16
17
18  /**
19   * NtpClient - an NTP client for Java.  This program connects to an NTP server
20   * and prints the response to the console.
21   *
22   * The local clock offset calculation is implemented according to the SNTP
23   * algorithm specified in RFC 2030.
24   *
25   * Note that on windows platforms, the curent time-of-day timestamp is limited
26   * to an resolution of 10ms and adversely affects the accuracy of the results.
27   *
28   *
29   * This code is copyright (c) Adam Buckley 2004
30   *
31   * This program is free software; you can redistribute it and/or modify it
32   * under the terms of the GNU General Public License as published by the Free
33   * Software Foundation; either version 2 of the License, or (at your option)
34   * any later version.  A HTML version of the GNU General Public License can be
35   * seen at http://www.gnu.org/licenses/gpl.html
36   *
37   * This program is distributed in the hope that it will be useful, but WITHOUT
38   * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
39   * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for
40   * more details.
41   *
42   * @author Adam Buckley
43   */
```

```
44 public class SntpClient
45 {
46 /**
47  * modificação
48  *************/
49 //public static void main(String[] args) throws IOException
50 //{
51
52    String serverName;
53
54    public SntpClient(String servidor) {
55       serverName = servidor;
56    }
57
58    public Date informaData() throws IOException {
59
60
61    /**
62     * modificação
63     *************/
64    // Process command-line args
65    /*if(args.length==1)
66      {
67         serverName = args[0];
68      }
69      else
70      {
71         printUsage();
72         return;
73      }*/
74
75    // Send request
76    DatagramSocket socket = new DatagramSocket();
77    InetAddress address = InetAddress.getByName(serverName);
78    byte[] buf = new NtpMessage().toByteArray();
79    DatagramPacket packet =
       new DatagramPacket(buf, buf.length, address, 123);
80
81    // Set the transmit timestamp *just* before sending the packet
82    // ToDo: Does this actually improve performance or not?
83    NtpMessage.encodeTimestamp(packet.getData(), 40,
       (System.currentTimeMillis()/1000.0) + 2208988800.0);
84
85    socket.send(packet);
86
```

```
87
88      // Get response
89      /**
90       * modificação
91       *************/
92      //System.out.println("NTP request sent, waiting for response...\n");
93      packet = new DatagramPacket(buf, buf.length);
94      socket.receive(packet);
95
96      // Immediately record the incoming timestamp
97      double destinationTimestamp =
         (System.currentTimeMillis()/1000.0) + 2208988800.0;
98
99
100     // Process response
101     NtpMessage msg = new NtpMessage(packet.getData());
102
103     // Corrected, according to RFC2030 errata
104     double roundTripDelay = (destinationTimestamp-msg.originateTimestamp) -
         (msg.transmitTimestamp-msg.receiveTimestamp);
105
106     double localClockOffset =
         ((msg.receiveTimestamp - msg.originateTimestamp) +
          (msg.transmitTimestamp - destinationTimestamp)) / 2;
107
108
109     // Display response
110     /**
111      * modificação
112      *************/
113     /*System.out.println("NTP server: " + serverName);
114       System.out.println(msg.toString());
115
116       System.out.println("Dest. timestamp:     " +
           NtpMessage.timestampToString(destinationTimestamp));
117
118       System.out.println("Round-trip delay: " +
           new DecimalFormat("0.00").format(roundTripDelay*1000) + " ms");
119
120       System.out.println("Local clock offset: " +
121        new DecimalFormat("0.00").format(localClockOffset*1000) + " ms");*/
122
123     socket.close();
124
125     /**
```

```
126     * modificação
127     *************/
128    return msg.data(msg.receiveTimestamp);
129 }
130
131
132 /**
133  * Prints usage
134  */
135 static void printUsage()
136 {
137    System.out.println(
       "NtpClient - an NTP client for Java.\n" +
       "\n" +
       "This program connects to an NTP server and prints the response to the console.\n" +
       "\n" +
       "\n" +
       "Usage: java NtpClient server\n" +
       "\n" +
       "\n" +
       "This program is copyright (c) Adam Buckley 2004 and distributed under the terms\n" +
       "of the GNU General Public License.  This program is distributed in the hope\n" +
       "that it will be useful, but WITHOUT ANY WARRANTY; without even the implied\n" +
       "warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU\n" +
       "General Public License available at http://www.gnu.org/licenses/gpl.html for\n" +
       "more details.");
138
139 }
140 }
```

B.22 Verifica.java

```
1  package assina.Assinatura;
2
3  import java.io.*;
4  import java.util.*;
5  import java.util.Date;
6  import java.security.*;
7  import java.security.cert.*;
8  import java.security.cert.Certificate;
9  import org.bouncycastle.jce.PKCS7SignedData;
10 import org.bouncycastle.jce.provider.BouncyCastleProvider;
11 import assina.Arquivo.ArqByte;
```

```
12 import assina.Carimbo.CarimboTempo;
13
14 public class Verifica {
15
16    byte[] arquivo, resultado;
17    ArqByte objArquivo;
18    PKCS7SignedData pkcs7;
19
20    public Verifica() {
21    }
22
23    public boolean verificar(String arq) throws CRLException,
       InvalidKeyException, CertificateException, NoSuchProviderException,
        NoSuchAlgorithmException, SignatureException {
24       String[] s = arq.split(".pkcs7");
25       CarimboTempo carimbo = new CarimboTempo();
26       Timestamp time = carimbo.informarCarimboTempo(s[0] + ".tstamp");
27       Date dtCarimboTempo = time.getTimestamp();
28       File arqOriginal = new File(s[0]);
29       Date dtArqOrig = new Date(arqOriginal.lastModified());
30       if (dtCarimboTempo.compareTo(dtArqOrig) < 0) {
31          System.out.println("arquivo modificado depois do carimbo de tempo");
32          return false;
33       }
34       objArquivo  = new ArqByte();
35       try {
36          resultado = objArquivo.entrarDados(arq);
37          arquivo = objArquivo.entrarDados(s[0]);
38       } catch (Exception e) {
39          e.printStackTrace();
40       }
41       pkcs7 = new PKCS7SignedData(resultado);
42       try {
43          pkcs7.update(arquivo, 0, arquivo.length);
44          arquivo = null;
45          String data = time.getTimestamp().toString();
46          arquivo = data.getBytes();
47          pkcs7.update(arquivo, 0, arquivo.length);
48          return pkcs7.verify();
49       } catch (SignatureException se) {
50          se.printStackTrace();
51          return false;
52       }
53    }
54 }
```