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

Jeandré Monteiro Sutil

Implementando novas abordagens para a troca do par de chaves de Autoridades Certificadoras.

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

Marcelo Carlomagno Carlos
Orientador

Prof. Ricardo Felipe Custódio, Dr.
Co-Orientador

Florianópolis, julho de 2007

Implementando novas abordagens para a troca do par de chaves de Autoridades Certificadoras.

Jeandré Monteiro Sutil

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

Prof. José Mazzuco Júnior, Dr.

Coordenador do Curso

Banca Examinadora

Marcelo Carlomagno Carlos

Orientador

Fabiano Castro Pereira, M.Sc.

Prof. Júlio da Silva Dias, Dr.

No que diz respeito ao empenho, ao compromisso, ao esforço, à dedicação, não existe meio termo. Ou você faz uma coisa bem feita ou não faz.
Ayrton Senna da Silva

Aos meus pais, as duas pessoas que mais contribuíram para que eu chegasse até aqui.

Agradecimentos

Agradeço aos amigos que fizeram de tudo para que esse trabalho jamais fosse concluído, mas que tiveram um papel importantíssimo para que fosse mais fácil chegar ao seu fim. À minha namorada pela paciência nas inúmeras vezes em que eu estive hipnotizado em frente ao computador. À minha família que sempre apoiou minhas decisões e de onde tirei forças para enfrentar todo e qualquer obstáculo que encontrei pela frente. Um agradecimento especial ao Laboratório de Segurança em Computação da UFSC - LabSEC e a todos que o compõe, pois viabilizaram a execução desse trabalho e mostraram que um ambiente seguro pode coexistir com a descontração de um abiente acadêmico. *(Jeandré Monteiro Sutil)*

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

AC	Autoridade Certificadora
AC-Raiz	Autoridade Ceritificadora Raiz
AKID	Authority Key Identifier
AR	Autoridade de Registro
DPC	Declaração de Práticas de Certificação
ICP	Infra-estrutura de Chaves Públicas
ICP-Brasil	Infra-estrutura de Chaves Públicas Brasileira
LabSEC	Laboratório de Segurança em Computação - UFSC
LCR	Lista de Certificados Revogados
PC	Políticas de Certificação
PHP	Hypertext Preprocessor
PKCS#7	Public-Key Cryptography Standards #7
PKCS#12	Public-Key Cryptography Standards #12
RFC	Request For Comment
RSA	Rivest-Shamir-Adleman
SHA	Secure Hash Algorith
SKID	Subject Key Identifier
X.509	Padrão de gerência de certificados digitais

Resumo

A certificação digital tem se mostrado uma ferramenta fundamental na manutenção da segurança em ambientes digitais. Para contribuir com sua evolução e aumentar ainda mais sua aceitação, esse assunto merece ser detalhadamente estudado, com o intuito de garantir sua eficiência em qualquer ocasião, agregando assim cada vez mais confiabilidade e robustez.

Uma das situações mais sensíveis na vida útil de uma ICP é a troca do par de chaves de uma AC-Raiz. Esse trabalho aborda os procedimentos encontrados hoje e mostra a implementação de novas propostas para a renovação do par de chaves de uma AC-Raiz e a atualização de seu certificado, contribuindo para a manutenção da ICP a longo prazo. Ainda, uma proposta de renovação *on-line* de certificados digitais de entidades finais é apresentada para automatizar tal procedimento, simplificando o processo de reemissÃ£o do certificado, tanto para a AC, quanto para a entidade subordinada.

Palavras chave: Certificação Digital, Infra-estrutura de Chaves Públicas, Certificado Digital, Autoridade Certificadora.

Abstract

Digital Certification has become an important tool in computing security. To improve it, a detailed study is necessary to validate each of the aspects covered by the theme. One of the most harmful situations in the ICP life cycle, is the replacement of its key pair, over all in the case of Root-CA's. This document describes the procedures found into the literature to handle this kind of situation and shows new methods related to the replacement of the key pair and the update of Root CA's certificate. Furthermore, a tool was implemented to simplify the renewal of end users certificates.

Keywords: Digital Certification, Public Key Infrastructure, Digital Certificate, Certification Authority.

Capítulo 1

Introdução

O padrão X.509, proposto pelo ITU-T *International Telecomunication Union* [IT 98], prevê uma série de medidas necessárias à implementação e gerenciamento de infra-estruturas de chaves públicas e, conseqüentemente, de seus certificados digitais. Apesar de ser um padrão muito abrangente, sua especificação deixa um pouco a desejar quando se trata da troca do par de chaves, no caso de comprometimento da chave privada antiga. Por ser um evento de vital importância para a manutenibilidade e disponibilidade de uma Infra-estrutura de Chaves Públicas (ICP), deve ser estudado e abordado com mais detalhes, para que em uma situação como essa os custos sejam minimizados e a segurança da cadeia de certificação não seja comprometida.

1.1 Contextualização

Este trabalho está inserido no contexto de módulos públicos para infra-estrutura de chaves públicas. O módulo público é a interface entre o cliente (usuário) e os componentes autoridade certificadora (AC) e autoridade de registro (AR). Em particular, foram estudados os mecanismos que permitem ao usuário a substituição, de forma simples, do seu certificado por outro, emitido por uma AC pertencente à mesma árvore de certificação. Neste sentido, o presente trabalho faz parte do esforço do LabSEC em estudar e desenvolver ferramentas de módulo público.

1.2 Objetivos

Este trabalho tem por objetivo a implementação e validação de novas propostas para melhorar os protocolos e políticas que regem uma Infra-estrutura de Chaves Públicas, no que diz respeito à troca do par de chaves de autoridades certificadoras, bem como a alteração ou atualização de seus certificados.

1.3 Objetivos Específicos

- Implementar novos modelos para a troca do par de chaves de autoridades certificadoras e a atualização de seus certificados;
- Analisar o impacto dos novos modelos sobre todas as autoridades pertencentes à árvore de certificação;
- Validar um modelo de gerência de chaves onde o impacto da troca do par de chaves seja mínimo e previsível.

1.4 Justificativa

Com a validação das melhorias propostas, o grau de confiabilidade do protocolo descrito pelo padrão X.509 seria elevado, bem como em uma futura necessidade real dessa substituição do par de chaves ou do certificado, os transtornos de tal procedimento poderiam ser estimados e reduzidos com base nos resultados obtidos.

1.5 Metodologia

Para a realização do trabalho serão estudados os modelos relativos à troca do par de chaves de uma autoridade certificadora descritos no padrão X.509, além de novas abordagens para a solução de tal problema. Em seguida, serão implementados protótipos para simular a aplicação das novas propostas em uma ICP de teste, com

o intuito de analisar os resultados obtidos e compará-los aos dos métodos já existentes. Por fim será definido o modelo de maior abrangência e com o menor impacto no processo de troca de chaves, baseando-se na análise dos resultados obtidos.

1.6 Limitações do Trabalho

No presente trabalho não serão abordados aspectos relevantes ao processo de criação do par de chaves, além do impacto da geração do novo par de chaves em relação as políticas de certificação da AC. Quanto à simulação da ICP, não serão abordados todos os procedimentos de segurança aplicados a mesma, como a rigidez na verificação das requisições e protocolos de segurança que envolvem a emissão de certificados e criação de ACs. Parte-se do pressuposto de que a ICP já esteja operando em um ambiente seguro.

Capítulo 2

Fundamentos Criptográficos

A segurança em ambientes computacionais visa prevenir que possíveis vulnerabilidades inerentes a tais sistemas sejam exploradas, tornando o custo da tentativa proibitivo. Para tanto, uma das ferramentas amplamente usadas para a implementação da segurança em computação é a criptografia [PFL 89].

2.1 Criptografia

Segundo [SCH 93], “Criptografia é a arte e a ciência de manter mensagens seguras.”

Trazendo essa definição para o meio computacional, a criptografia pode ser entendida como sendo uma área que procura implementar segurança através da conversão de informações legíveis e inteligíveis em algo aparentemente sem sentido. Para tanto, são usadas funções matemáticas cuja finalidade é embaralhar a informação e torná-la tão imprevisível que a reversão o processo seja praticamente impossível (cifragem). A única maneira de se recuperar tais informações deve ser com o conhecimento de um segredo que possibilite a reversão da cifragem, trazendo de volta à informação seu significado original (decifragem).

O segredo no qual se baseiam a maior parte dos algoritmos criptográficos [NIS 77, NIS 94, NIS 01, TEC 02] é conhecido como chave criptográfica . Sendo as-

sim, para a cifragem de dados o usuário deve fornecer uma chave e para decifrá-los deve conhecer a chave correspondente, podendo esta ser a própria chave usada na cifragem ou uma outra chave capaz de reverter o efeito causado pela primeira, dependendo do tipo de criptografia empregada, como será visto a seguir.

O modo como são empregadas as chaves nos permitem dividir os conceitos criptográficos em:

- Funções Resumo;
- Criptografia de chave simétrica;
- Criptografia de chave assimétrica.

2.1.1 Funções Resumo

Funções resumo são usadas para sintetizar informações de tamanho qualquer em um resumo de tamanho fixo. Essas funções são tidas como não inversíveis, apesar de matematicamente não estar provado se podem ou não ser revertidas. De qualquer forma, podemos então dizer que são funções cuja aplicação é computacionalmente fácil e sua inversão bastante difícil [MEN 96].

Essas funções são freqëntemente usadas para garantir integridade de documentos, guardar senhas em arquivos, dentre muitas outras aplicações. Em princípio, esse tipo de função não requer uso de chave, sendo a informação a ser processada a sua única entrada. A característica mais forte e que dá o nome às funções resumo, é o fato de que não importa o tamanho do documento a ser cifrado, sua função resumo terá sempre o mesmo tamanho, se o mesmo algorítmo e parâmetros forem aplicados. Assim, dados cifrados por elas não podem, na prática, ser decifrados, pois há perda de informação no processo.

Uma função resumo ideal deve evitar colisões, ou seja, a partir de entradas diferentes produzir o mesma saída. Porém, é impossível a unicidade do par (entrada, saída), uma vez que há perda de informação, ou seja, o domínio da função resumo

é maior que seu contra-domínio e a repetição de resultados a partir de entradas diferentes é inevitável [SZY 05]. Contudo, a dificuldade está em encontrar informações diferentes que tenham, cada uma, um significado e produzir o mesmo resumo.

Outra situação que deve ser evitada é: a partir de um resumo e alterações nos dados que o geraram, chegar a outros dados significantes que gerem esse mesmo resumo. As funções de resumo devem tornar esse tipo de tentativa inviável.

2.1.2 Criptografia Simétrica

O processo criptográfico onde é usada a mesma chave para cifrar e decifrar mensagens é conhecido como criptografia simétrica. Tal chave é conhecida como segredo compartilhado, uma vez que deve ser previamente conhecida pelas partes envolvidas na comunicação. Em geral, algoritmos de criptografia simétrica exigem menos esforço computacional que os de criptografia assimétrica (veja 2.1.3). A figura 2.1 ilustra um exemplo de comunicação usando criptografia simétrica:

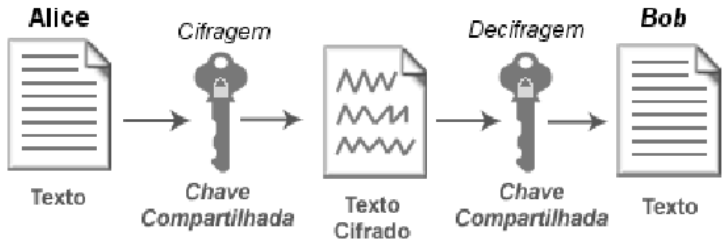

Figura 2.1: Figura representando comunicação com criptografia simétrica

Um bom algoritmo simétrico deve impedir que uma terceira pessoa não autorizada encontre algum padrão na mensagem cifrada que a leve ao texto original, usando por exemplo um ataque de análise de freqüência, onde o atacante mapeia a quantidade de cada caracter presente na mensagem e faz uma comparação com as letras mais frequentes em textos de um determinado idioma. Ele deve também tornar impossível que o atacante, a partir do texto cifrado, descubra a chave que o cifrou.

Por seu determinismo, algoritmos criptográficos estão sempre sujeitos a ataques de força bruta, onde tenta-se encontrar a chave necessária para a decifragem da mensagem explorando todas as possibilidades. Porém para uma boa chave, esse ataque torna-se inviável, uma vez que pode levar anos, ou até séculos para explorar todo o domínio de entradas. Com isso, fica evidente a importância da escolha de uma boa chave (o mais imprevisível possível) para a segurança da comunicação [SCH 93].

Um ponto crítico na criptografia simétrica diz respeito ao compartilhamento de chaves, pois uma interceptação das mesmas no momento da distribuição comprometeria toda a segurança na comunicação. Esse impasse foi solucionado com o uso da criptografia assimétrica, descrita a seguir. Outro problema é a necessidade de uma chave para cada troca segura de mensagens que se fizer necessária. Um equívoco poderia permitir que a mensagem fosse entregue ao destino errado e esse tivesse a possibilidade de decifrá-la, o que, dependendo da ocasião, é indesejável [HOU 01].

2.1.3 Criptografia Assimétrica

A criptografia assimétrica funciona de maneira que duas chaves estão envolvidas no processo de cifragem/decifragem. Essas duas chaves possuem uma ligação, sendo que dados cifrados por uma delas poderão ser decifrados somente pela outra. Porém a partir da primeira delas não é possível determinar a segunda, ou vice-versa.

Esse tipo de criptografia é também conhecido como criptografia de chaves públicas [SAL 96], uma vez que na prática o que ocorre nesse procedimento é que uma das chaves é mantida em segredo (chave privada), enquanto a outra é distribuída livremente (chave pública). A chave privada de uma entidade é usada para assinar documentos, provendo uma prova de autoria do mesmo. A chave pública, por sua vez, é usada para que outras entidades possam verificar a assinatura de tal documento, garantindo sua autenticidade, além de cifrar dados sigilosos para que somente o mantenedor da chave privada possa ter acesso à informação em questão.

Por ser computacionalmente mais cara, a criptografia assimétrica geralmente é usada em conjunto com a simétrica (já consolidada na cifragem de documentos

e comunicações), agindo justamente onde esta é mais fraca, ou seja, no compartilhamento do segredo, bem como na manutenção das chaves.

A figura 2.2 mostra uma possível aplicação da criptografia assimétrica na troca de mensagens eletrônicas:

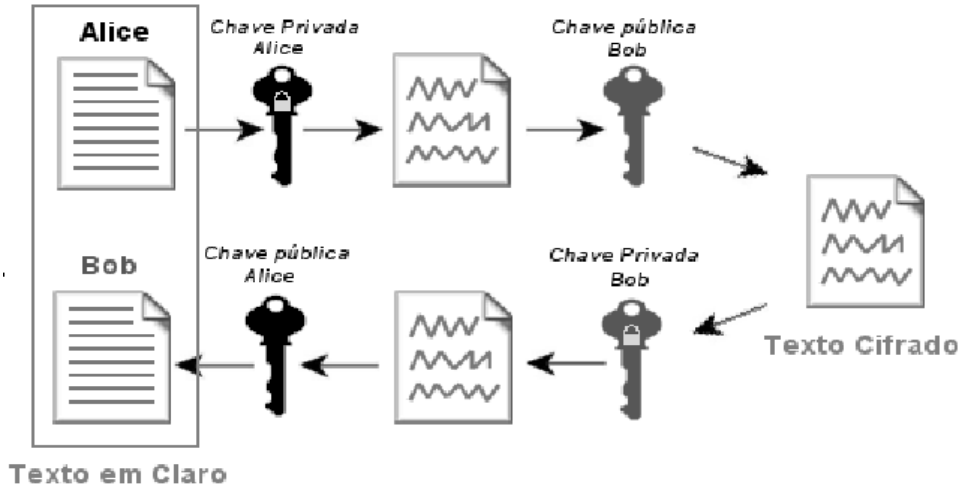

Figura 2.2: Figura representando comunicação usando criptografia assimétrica

2.2 Assinatura Digital

A assinatura digital é o instrumento usado com o principal objetivo de garantir autoria ou concordância, integridade e autenticidade de dados digitais e, por conseqüência, das informações agregadas a eles. Segundo [PFL 89] e [SCH 93] a assinatura digital ideal deve possuir as seguintes propriedades:

- Deve ser impossível de ser forjada;
- Deve prover autenticidade. A parte que recebe um documento deve ter a certeza de sua origem;
- Deve impedir que o conteúdo do documento possa ser alterado após sua assinatura;

- Não deve ser reusável. A assinatura de um documento não pode ser “copiada” para outro;
- Deve impedir o repúdio, ou seja, não deve permitir que a entidade que assinou determinado documento volte atrás e negue tal ação.

2.2.1 Assinatura Digital em Documentos Eletrônicos

Para assinar digitalmente um documento, o mesmo deve ser submetido a uma função resumo. O resumo resultante da função é então cifrado com a chave privada do assinante e concatenado ao documento. [NIS 92]

Para validar a assinatura do documento, qualquer pessoa que possua e confie na chave pública do assinante poderá usá-la para decifrar a assinatura concatenada ao documento, recuperando o resumo do mesmo. Com a aplicação da mesma função resumo usada anteriormente para a geração da assinatura, os dois resumos podem ser conferidos e caso sejam idênticos, fica confirmada a autoria da mensagem.

Capítulo 3

Infra-estrutura de Chaves Públicas

Uma das formas de se garantir segurança em transações eletrônicas é através da implantação de uma Infra-estrutura de Chaves Públicas (ICP). Com o uso da certificação digital, é possível proporcionar aos usuários autenticidade, integridade, sigilo e não-repúdio em um ambiente inseguro, possibilitando assim a execução de operações que necessitem de um grau elevado de segurança, como transações bancárias, envio de emails seguros e sigilosos, acesso a locais restritos, entre outras aplicações.

Para tanto, a ICP conta com uma série de componentes que desempenham papéis distintos necessários à sua implantação e manutenção. Os principais componentes serão expostos no decorrer do capítulo.

Há várias arquiteturas possíveis para uma ICP, dentre elas: hierárquica, rede de confiança, híbrida, de ponte, etre outras. O foco deste documento será voltado às ICPs hierárquicas, contudo, informações sobre as demais abordagens podem ser encontradas em [HOU 01] e [ADA 00].

3.1 Padrão X.509

O padrão X.509 é a especificação de práticas relacionadas à Infra-estrutura de Chaves Públicas mais difundida na atualidade. Nele estão contidas informações que abrangem estruturas, protocolos, práticas e sugestões de uso, dentre muitos outros

aspectos relacionados à utilização da certificação digital. Isso faz com que o X.509 seja um padrão muito extenso e abrangente, estando distribuído hoje em mais de trinta RFCs, que podem ser consultadas em `http://www.ietf.org/rfc`, além do documento no qual foi proposto [IT 98].

3.1.1 Certificados Digitais

Um certificado digital é um objeto eletrônico capaz de identificar uma entidade através de seus dados, sua chave pública e a assinatura de uma terceira parte confiável. Para tanto, agrega funcionalidades de dois objetos reais bastante comuns, o cartão de crédito e o cartão de visitas [HOU 01].

Em um cartão de visitas estão todas as informações pertinentes necessárias à identificação de uma certa entidade. Nele geralmente estão contidas informações como nome, telefone, empresa onde trabalha (no caso de um cartão profissional), e-mail, endereço, cargo exercido, entre outras informações. O dono do cartão pode até colocar a sua chave pública em seu verso, com o intuito de trocar mensagens eletrônicas seguras.

Porém, nada garante que em um primeiro momento, duas pessoas que acabaram de se conhecer possam confiar nos cartões uma da outra. Uma pessoa mal intencionada pode mandar fazer cartões com o nome da pessoa cuja identidade deseja forjar e colocar no verso a sua própria chave pública. Até que a fraude seja descoberta, muitos danos poderão já ter sido causados. É para evitar esse tipo de problema que o certificado digital agrega também funcionalidades de um cartão de crédito.

O cartão de crédito comum tem algumas propriedades interessantes que estão agregadas no certificado digital. Um cartão de crédito é geralmente difícil de ser falsificado, uma vez que possui os dados de seu titular em alto relevo, hologramas para dificultar a clonagem e devem ainda conter em seu verso um espaço para a assinatura do titular, como forma de conferir a identidade da pessoa que efetua a transação. Além dessas informações o cartão ainda conta com uma data de validade que busca assegurar a atualidade das demais informações. Esses dados presentes em um cartão de crédito

são assegurados por uma terceira parte confiável, que seria a empresa que emite e gerencia os mesmos. Outra característica fundamental do cartão de crédito é que ele possui um limite, ou seja, uma restrição de quanto cada titular pode usufruir desse recurso. No caso desse limite ser extrapolado, o cartão é bloqueado. O bloqueio também é possível caso o titular informe a perda, extravio ou até mesmo furto do cartão, ou ainda caso a validade tenha expirado.

Todas as características expostas são importantes em um certificado digital. Fazendo uma síntese das propriedades presentes nos objetos descritos acima, temos idéia das informações presentes em tais certificados, como mostrado na tabela 3.1.

Um certificado digital ideal deve possuir as seguintes propriedades [HOU 01] :

- deve ser totalmente digital para facilitar a sua distribuição e processamento no meio virtual;
- deve conter as informações necessárias para se identificar seu dono (portador da chave privada);
- deve ser fácil de determinar a validade do mesmo, ou seja, deve possuir a data de emissão e validade;
- deve ter sido criado por uma terceira parte confiável e não pelo próprio dono do certificado;
- deve ser fácil de diferenciar dois ou mais certificados de uma mesma identidade, tendo em vista que não há restrição de número de certificados por entidade;
- deve ser fácil de identificar uma tentativa de adulteração de um certificado;
- não deve ser possível alterar seu conteúdo;
- deve discriminar o tipo de aplicação para o qual foi designado.

	Cartão Pessoal	**Cartão de Crédito**	**Certificado Digital**
Nome ou razão social da entidade	Presente	Presente	Compõe o campo *subject*
Outros dados do titular (CPF, CNPJ, RG, cargo, pseudônimo, etc.)	Presente	Ausente	Presentes como *subject* ou na forma de extensões
Contato (endereço eletrônico, fone)	Presente	Ausente	Presentes como *subject* ou na forma de extensões
Endereço	Presente	Ausente	Presentes como *subject* ou na forma de extensões
Validade	Ausente	Presente	Inclusa nos campos *notAfter* e *notBefore* do certificado
Limitação de uso	Ausente	Presente	Presente como extensões de uso (*KeyUsage*)
Garantia de um terceiro confiável	Ausente	Presente	O certificado digital é emitido e assinado por uma terceira parte confiável (AC)
Possibilidade de revogação	Ausente	Presente	Um certificado pode ser revogado por seu emissor. Uma lista com os certificados que se encontram nessa condição é emitida e divulgada periodicamente pela AC

Tabela 3.1: Tabela ilustrando a composição do certificado digital com características bem conhecidas em cartões pessoais e cartões de crédito

3.1.2 Requisição de Certificado

Quando um novo certificado precisa ser emitido, é necessário que uma requisição de certificado, também conhecida com requisição de assinatura de certificado, seja criada. Nessa requisição estão os dados necessários para identificar o portador de determinada chave privada e a chave pública correspondente, que deverá ser incorporada ao certificado. As requisições podem ainda conter alguns campos adicionais, na forma de extensões opcionais [NYS 00]. Dependendo da política de certificação adotada, esses campos opcionais podem variar desde atributos adicionais para melhor identificar a entidade requisitante, o propósito a que se destina o certificado como até mesmo uma senha que será usada para efetuar uma possível revogação do certificado. Questões envolvendo política de certificação serão vistas a seguir em 3.1.5.

3.1.3 Autoridade Certificadora

A autoridade certificadora em uma ICP é a entidade responsável por emitir os certificados digitais tanto de outras autoridades como de entidades finais. Em outras palavras, a AC recebe uma requisição para emissão de um certificado que, se aprovada, é utilizada na a construção de um novo certificado digital. Esse certificado digital é então assinado com a chave privada da autoridade certificadora e entregue ao requisitante. [ADA 02]

Além de emitir certificados, a AC é responsável também por supervisionar esses certificados durante toda a sua vida útil. Isso significa que caso ocorra algum problema com a chave privada correspondente a um certificado emitido pela AC ou mesmo a necessidade de mudança em algum dos atributos do certificado, a mesma deverá revogar o certificado antigo.

Para que outros usuários fiquem cientes dos certificados que foram revogados, a AC deve publicar uma lista com informações das suas revogações, conhecida como Lista de Certificados Revogados ou LCR, que será abordada na seção 3.1.4.

É importante também que sejam mantidos os históricos dos certificados

e LCRs emitidos, bem como outras atividades desempenhadas pela AC caso seja necessária uma auditoria. [HOU 01]

Esse conjunto de atribuições fazem da autoridade certificadora uma entidade de suma importância em uma infra-estrutura de chaves públicas. Por ser o ponto de confiança comum a todos os usuários e autoridades subordinadas, a segurança aplicada à proteção da chave privada da AC deve ser inviolável, pois uma vez comprometida, toda a cadeia de certificação estará em risco. Essa preocupação é ainda maior quando se trata de uma autoridade certificadora raiz, o ponto máximo de confiança de uma ICP.

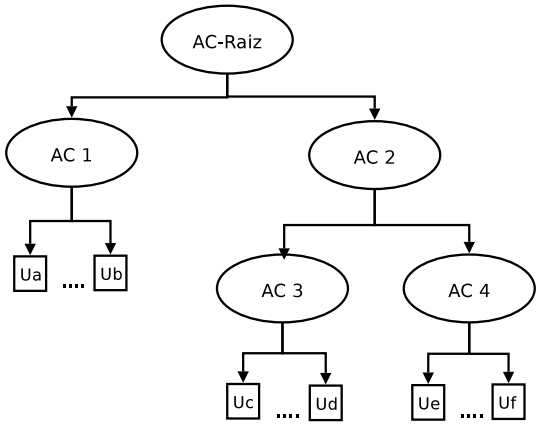

Figura 3.1: Uma ICP hierárquica. A AC-Raiz (no topo), suas subordinadas (AC i ... AC n) e os certificados de usuários finais (Ui ... Un)

Uma autoridade certificadora raiz é a base de toda a árvore de certificação em uma ICP. Seu certificado é auto-assinado, ou seja, é emitido pela própria AC e assinado por sua própria chave privada. Na prática, as chaves privadas de autoridades (sobretudo da AC-Raiz), em grandes ICPs, são armazenadas em salas cofre que, como o nome já diz, asseguram a segurança física da chave. Porém, para assegurar que uma pessoa com acesso a essa área restrita faça uma cópia da chave ou tenha qualquer tipo de contato com a mesma, a chave privada de uma AC é armazenada em um Módulo de segurança Criptográfico ou MSC.

O MSC é um hardware construído para gerenciar o ciclo de vida de um ou mais pares de chaves. Dentro dele a chave é criada, utilizada e nele deve permanecer até expirar (no caso de haver um limite de uso) ou até que o MSC seja destruído. Para proteger a(s) chave(s), o MSC mantém um determinado perímetro que, uma vez violado, provoca a destruição da(s) mesma(s). Quando um documento precisa ser assinado, como por exemplo um certificado, o sistema responsável pelo gerenciamento de certificados submete um resumo dos dados para que seja assinado pela chave privada contida no MSC e recebe de volta o resumo assinado. Dessa maneira não há um contato direto entre chave privada e o exterior do hardware. Mais informações sobre o uso de MSCs podem ser encontradas em [MAR 05].

3.1.4 Lista de Certificados Revogados

Para manter as informações de uma determinada AC atualizadas em caso de perda ou roubo de chaves privadas, extravio, ou qualquer outro caso onde os mantenedores dessas chaves ficassem impossibilitados de usá-las, se fez necessária uma forma de informar tal ocorrência a todos os usuários da ICP [MAR 05].

Para tanto, foi criada a Lista de Certificados Revogados, uma lista contendo o registro de todos os certificados revogados por uma AC que ainda não perderam a validade. Essa lista é publicada de forma que qualquer usuário da ICP possa conferir a validade de um determinado certificado através de sua obenção e posterior verificação. A lista contém informações como serial, data de revogação e opcionalmente um conjunto de extensões com informações adicionais de cada certificado revogado. Na prática é usada a extensão *Reason Code*, com o código que descreve a razão da revogação do certificado [HOU 02].

Continuando com a idéia do cartão de crédito, a LCR serviria então como as chamadas “Listas Negras” das operadoras, contendo os números de cartões de créditos com alguma restrição, que devem ser rejeitados.

3.1.5 Política de Certificação (PC)

As políticas de certificação definem um conjunto de regras sobre uma ICP que devem ser cumpridas pelas entidades credenciadas. Dentre as questões abordadas em uma política de certificação, podemos citar o protocolo de segurança que cada AC deverá cumprir na emissão de certificados ou credenciamento de ACs, a validade padrão para os certificados, entre outros. Assim, é através da análise dos documentos de políticas que terceiras partes decidem se um certificado é confiável e aplicável.

Como mencionado, as políticas são definidas em documentos - Políticas de Certificação (PC) e Declaração de Práticas de Certificação (DPC). A PC é um documento de alto nível em que se definem as práticas de segurança para emissão de certificados bem como o mantenimento das informações de certificado [HOU 01]. Descrevem-se, dessa maneira, a operação de uma AC, como ainda as responsabilidades dos usuários ao solicitar, usar e manipular certificados. Cabe à Declaração de Práticas de Cerficação - documento altamente detalhado - descrever como uma determinada AC implementa as regras definidas na PC, ou seja, como funcionará o protocolo de solicitação de certificados, descrito acima, por exemplo.

Portanto, a DPC é o documento que deve ser analisado por auditores para se validarem as operações de uma AC levando-se em consideração uma PC. Esta deve ser genérica e seu uso estimado por vários anos, enquanto aquela deve ser especificada para cada AC, sendo bem mais específica. Por último, a PC deve ser públicada, sendo que o mesmo não é necessário para a DPC.

3.1.6 Caminho de Certificação

Antes que o certificado de terceiros possa ser utilizado, deve haver uma verificação para garantir que o mesmo é digno de confiança. Essa verificação baseia-se na tentativa de se estabelecer uma conexão entre o certificado desconhecido e um ponto de confiança já estabelecido. Para isso, cada certificado entre esses dois pontos devem ser verificados [LOI 02].

Apesar de haver muitas dissidências com relação a montagem do caminho

de certificação e pouca documentação disponível sobre o assunto [LOI 02], há alguns passos que devem ser seguidos para sua validação, de acordo com o padrão X.509 [HOU 02]:

- Construção do caminho de certificação
- Validação do caminho de certificação

Contrução do Caminho de Certificação

Nessa fase tenta-se estabelecer um caminho "candidato" entre o ponto de confiança e o certificado a ser validado. Alguns passos devem ser seguidos para a construção do caminho, como veremos a seguir.

A forma mais básica de construção de um caminho de certificação é o encadeamento de nomes, que consiste na comparação do campo *subject* do emissor com o campo *issuer* do certificado em questão. Tal procedimento é repetido até se atingir um ponto de confiança (certificado auto-assinado) ou se esgotarem as possibilidades. Esse procedimento é mostrado na figura 3.2

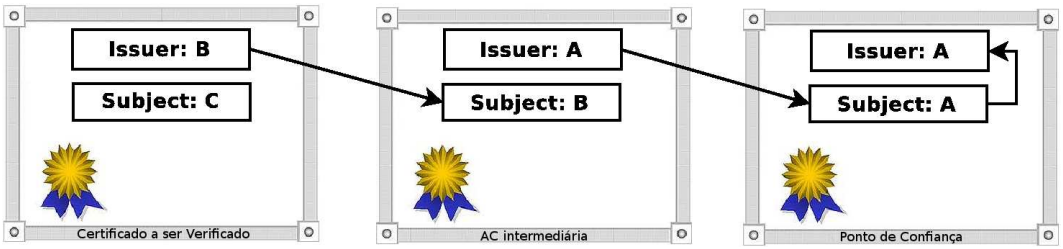

Figura 3.2: Construção do caminho de certificação baseado no encadeamento de nomes

Com o amadurecimento do padrão X.509, viu-se que o método acima poderia ser mais eficiente, visto que uma entidade pode possuir mais de um par de chaves, tendo assim dois certificados. Nesse caso, se os certificados contiverem o mesmo *subject*, haverá dois caminhos candidatos possíveis, ainda que somente um deles seja o caminho correto. Seria interessante então uma forma de se descartar logo na construção

esses caminhos, para melhorar o desempenho e diminuir o custo do processo. Para tanto, se fez necessária a adição de um novo método para a construção do caminho que identificasse a chave contida no certificado, não o próprio sujeito. Assim, com a terceira versão de certificados X.509, o mesmo passou a permitir duas extensões com esse fim, são elas: *Subject Key Identifier* (SKID) e *Authority Key Identifier* (AKID).

A figura 3.3 ilustra o processo envolvido na construção do caminho de certificação, quando as duas extensões estão disponíveis. O primeiro certificado possui o AKID completo, com serial e emissor. Já o segundo certificado possui apenas o identificador da chave. No terceiro, auto-assinado, a extensão foi omitida. As linhas tracejadas indicam que o par (serial,emissor) na extensão AKID são usados apenas para dar preferencia a um certo certificado dentre outros, não sendo necessário que tais campos combinem com os respectivos no certificado do emissor.

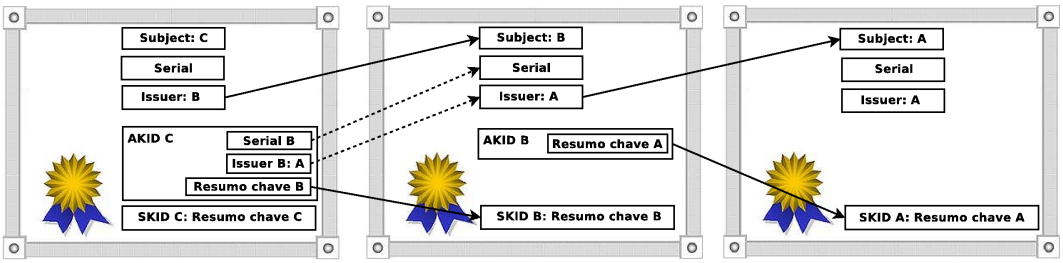

Figura 3.3: Construção do caminho de certificação utilizando as extensões AKID e SKID

A extensão *SubjectKeyIdentifier* (Identificador de chave do Sujeito) contém um identificador único para a chave pública. Geralmente esse identificador é um resumo da chave, com a aplicação das funções mostradas em 2.1.1. Porém há outros meios de produzir um identificador único para a chave, que fogem ao escopo desse documento, mas podem ser consultados em [LOI 02].

Segundo [HOU 02], a extensão *SubjectKeyIdentifier* deve estar presente em todos os certificados de autoridades certificadoras.

A extensão *AuthorityKeyIdentifier* (Identificador de chave da Autoridade) é geralmente composta pelo resumo da chave da AC emissora, podendo também con-

ter o serial do certificado emissor, bem como o campo *issuer* do certificado da AC emissora. Os dois últimos atributos servem apenas para dar preferência no caso da AC possuir mais de um certificado com o mesmo par de chaves. Um exemplo dessa situação é a renovação do certificado da AC, que será abordado com mais detalhadamente em 4.2.2. Segundo [HOU 02], essa extensão deve estar presente em todos os certificados emitidos por autoridades certificadoras, salvo quando o certificado for auto-assinado, ou seja, pertencente à própria autoridade que o emitiu, caso em que o *AuthorityKeyIdentifier* poderá ser omitido.

Com relação ao sentido na construção do caminho de certificação, duas técnicas podem ser abordadas. Uma delas é partir do certificado a ser validado tentando-se atingir um ponto de confiança. Essa técnica é a mais indicada para ICPs com arquitetura hierárquica, como a definida pelo padrão X.509. Outra técnica é partir dos pontos de confiança tentando alcançar o certificado a ser validado. É uma técnica que melhor se aplica a ICPs com arquitetura distribuída. Um bom algoritmo, no entanto, deve suportar ambos os tipos de montagem (direta ou reversa), usando ou não as extensões vistas acima.

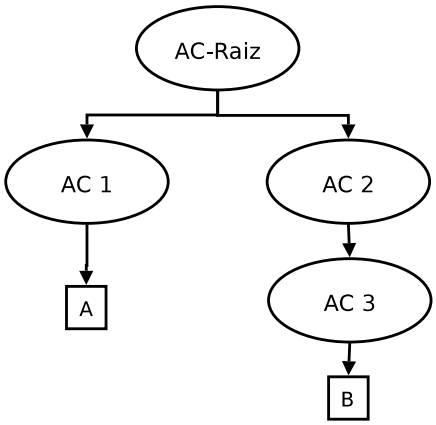

Figura 3.4: Um Caminho de certificação em uma ICP envolvendo A e B

A figura 3.4 mostra um caminho de certificação entre os certificados A e B. Para que o caminho seja montado, os certificados das ACs 1, 2, 3 e da AC-Raiz

devem estar na lista de certificados confiáveis das entidades A e B.

Validação do Caminho de Certificação

Na fase de validação, os caminhos candidatos são avaliados de acordo com a validade dos certificados envolvidos, a análise das listas de certificados revogados, assinaturas, restrições de uso e políticas de certificação [HOU 01].

Após essa verificação, caso algum dos candidatos passe nos testes, é então aceito e o certificado pode finalmente ser adicionado como um certificado confiável. Caso não haja um candidato aprovado, o certificado deverá ser rejeitado, pois não há nenhuma garantia de que seja um certificado confiável.

3.2 Infra-estrutura de Chaves Públicas Brasileira (ICP-Brasil)

De acordo com [ICP b], a ICP-Brasil é um conjunto de técnicas, práticas e procedimentos, a ser implementado pelas organizações governamentais e privadas brasileiras com o objetivo de estabelecer os fundamentos técnicos e metodológicos de um sistema de certificação digital baseado em chave pública.

A Infra-estrutura de chaves públicas brasileira é a única ICP com validade legal em território nacional, o que a torna de vital importância tanto para o setor público (em sites de serviços como o da recita federal), quanto em setores privados onde se faz necessário o uso de certificação digital para transações seguras (sites de bancos, autenticação em servidores).

Com o crescimento da disponibilidade e da utilização de serviços envolvendo transações financeiras online, é de suma importância uma forma de garantir confiança entre o usuário e o prestador de serviço, uma vez que com esse aumento na demanda, cresce também o número de ataques visando à obtenção de lucro de forma ilícita a partir da exploração de falhas em tais sistemas [ONL].

A estrutura da ICP-Brasil utilizada no trabalho é mostrada na figura 3.5,

com base na árvore de certificação formada por todas as autoridades certificadoras credenciadas até 13 de março de 2007 [ICP a]. As ACs em credenciamento foram omitidas por motivo de simplificação.

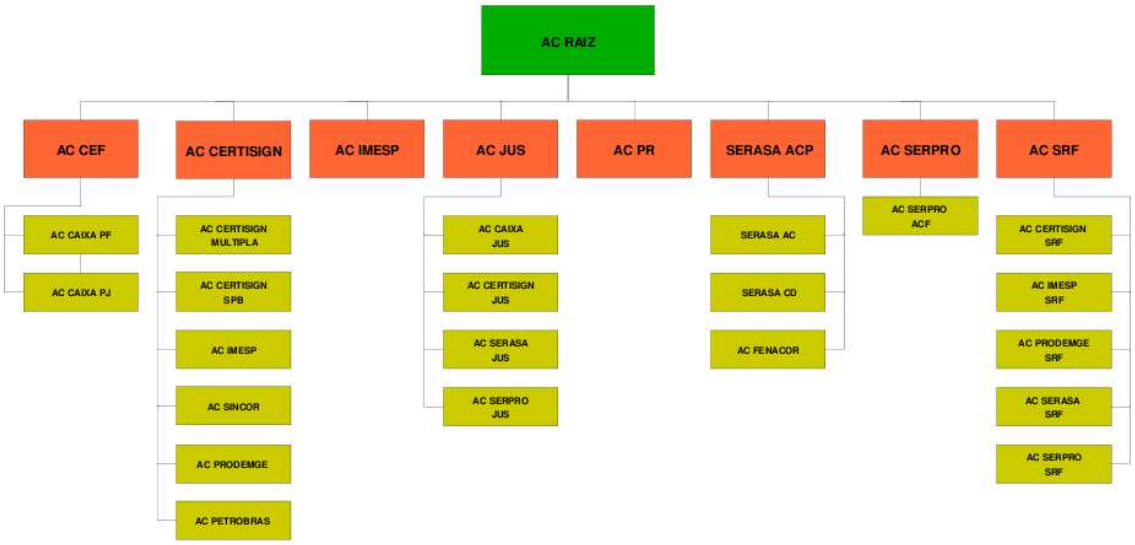

Figura 3.5: Hierarquia de autoridades certificadoras filiadas à ICP-Brasil

Capítulo 4

Propostas de Troca do Par de Chaves e Renovação de Certificado de Autoridades Certificadoras

4.1 Propostas encontradas na Literatura

Há na literatura duas propostas para a troca do certificado digital de uma autoridade certificadora [ADA 05, FRE 98]. Em ambos os casos o par de chaves antigo da AC deve estar disponível, como será exposto a seguir.

4.1.1 Protocolo de Gerenciamento de Certificado (CMP)

Para a troca do par de chaves de uma AC-Raiz, no caso do par de chaves antigo não estar comprometido, deverão ser emitidos certificados de transição [ADA 05]. Esse método envolve então a emissão de dois outros certificados adicionais, seguindo os seguintes passos:

- Um novo par de chaves deverá ser gerado para a AC e, com isso, um novo certificado auto-assinado deverá ser emitido com a utilização desse par (NCN);
- Um certificado de transição deverá ser emitido pela nova AC, para a AC antiga,

contendo assim a chave pública do certificado antigo da AC e sendo assinado por sua nova chave privada (ACN);

- Um segundo certificado de transição deverá ser emitido pela AC antiga, deverá conter a chave pública da nova AC, sendo assinado então pela chave privada antiga (NCA);

- O par de chaves antigo da AC não deverá mais ser usado, porém seu certificado antigo (ACA) deverá ser mantido até que perca a validade ou até que todos os certificados que o têm como ponto de confiança expirem.

Após a certificação cruzada, uma situação semelhante à da figura 4.1 deverá ocorrer.

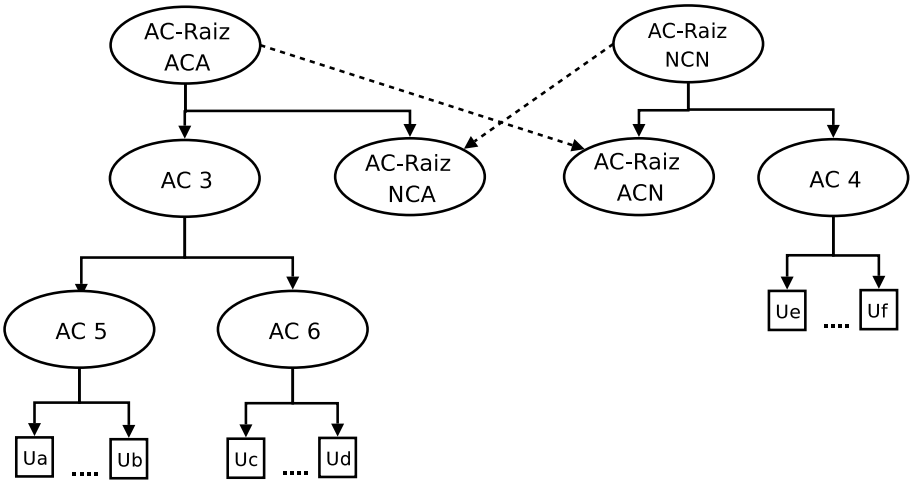

Figura 4.1: Hierarquia após aplicação do protocolo CMP

A figura 4.1 mostra a certificação cruzada entre a AC-Raiz antiga (ACA) e a nova AC-Raiz (NCN). Nesse processo outros dois certificados são criados, um com a chave pública antiga da AC-Raiz assinado pela chave nova (ACN) e outro com a chave pública nova da AC-Raiz assinado pela antiga (NCA).

Certificado antigo assinado com chave privada da nova AC (ACN)

Esse certificado de transição permite que os novos certificados assinados pela nova chave privada da AC estabeleçam um caminho de certificação até os certificados emitidos pela chave privada antiga da AC. A validade desse certificado deve partir da data de emissão do mesmo até a data de validade do certificado antigo da AC.

Certificado novo assinado com chave privada da AC antiga (NCA)

Com esse certificado de transição, os certificados assinados pela chave privada antiga da AC reconhecerão certificados assinados por sua nova chave privada. O prazo de validade deve ir da data de emissão até a data em que o último certificado assinado com a chave privada antiga da AC tenha expirado. No pior caso essa data será a data de expiração do certificado antigo da AC.

4.1.2 Lista de Certificados Confiáveis (CTL)

Trata-se de uma outra maneira de se resolver o problema apresentado em 4.1.1, mas que pode ser usado também para ACs intermediárias. Para a substituição da AC antiga por uma nova, a proposta é que a AC antiga emita uma lista de certificados confiáveis contendo uma referência para o certificado da nova [FRE 98].

Uma lista de certificados confiáveis é uma estrutura assinada utilizando o formato PKCS #7 [TEC 93] cuja pricipal função é a de listar um conjunto de certificados em que uma determinada AC confia. A lista foi criada para auxiliar a certificação cruzada, onde uma autoridade poderia emitir a lista contendo identificadores dos certificados pertencentes a outras ACs em que a primeira confiasse [CAR 07].

Dessa maneira, os certificados emitidos pela AC antiga confiariam em seu certificado e em sua CTL (do inglês *Certification Trust List*), uma vez que a lista também é assinada pela chave privada da AC. Como os certificados emitidos a partir da nova AC já passam a confiar na mesma, a CTL seria necessária até a expiração do último certificado que confiasse na AC antiga. No pior caso será necessária até a

data de expiração de seu próprio certificado.

Na figura 4.2, podemos visualizar a situação exposta acima, onde os certificados emitidos pela AC 1 passam a confiar na AC 2 a partir da emissão de uma CTL pela primeira identificando a segunda como confiável.

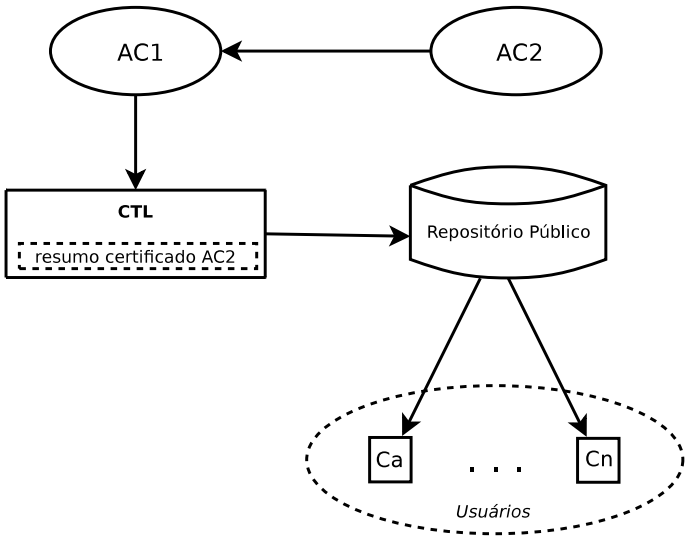

Figura 4.2: Substituição do par de chaves de uma AC-Raiz com uso de uma CTL

4.2 Novas Propostas

Em caso de comprometimento da chave privada da AC, os procedimentos descritos acima não são aplicáveis, pois todos utilizam o par de chaves antigo para emitir um novo certificado. Além disso, quando uma AC precisar mudar algum de seus atributos, por mudança em sua política, ou mesmo quando seu certificado expirar, um novo certificado deverá ser emitido, mantendo seu par de chaves antigo. Nesta seção serão expostas novas propostas para migrar a infra-estrutura de chaves públicas de um ponto de confiança para outro visando a minimização do custo da transição.

4.2.1 Troca do par de chaves sem chave privada antiga

Quando a chave privada de uma AC-Raiz não puder mais ser utilizada, nenhuma das duas abordagens expostas em 4.1.1 e 4.1.2 podem ser aplicadas, pois estes casos utilizam o par antigo para validar o novo par de chaves da AC.

No caso de comprometimento da chave privada de uma AC-Raiz, a única solução até agora era reemitir todos os certificados de uma ICP assinados com o novo par de chaves da AC, além de revogar o certificado antigo da raiz. [HOU 02]

Em uma ICP de grande porte, como é o caso da ICP-Brasil, esse alto custo teria um impacto catastrófico, pois significaria a reemissão de milhares ou até mesmo milhões de certificados. Segundo o Instituto Nacional de Tecnologia da Informação (ITI), a ICP-Brasil possuía em 2006 cerca de 500 mil certificados emitidos. A previsão para o fim de 2006 era de 1 milhão de certificados, com o crescente aumento da demanda [JAN].

Para evitar situações como essa, uma nova solução foi proposta, onde esse custo seria drasticamente reduzido [CAR to]. Segundo essa nova proposta, uma nova autoridade certificadora deverá ser criada para substituir a antiga AC-Raiz, cuja chave foi perdida. Para cada AC diretamente subordinada à antiga, uma nova requisição deve ser emitida e submetida à nova raiz. As requisições devem ser fiéis aos certificados assinados pela AC antiga. Assim, cada AC subordinada à AC antiga será também subordinada à nova AC.

Após o procedimento, os usuários da ICP precisarão apenas adicionar os novos certificados como ponto de confiança e o caminho de certificação será montado, mas agora passando pelo novo certificado da AC-raiz e suas subordinadas, conforme ilustrado na figura 4.3.

O certificado da AC antiga , bem como o de suas subordinadas, devem ser mantidos desde que se tenha a certeza de que o par de chaves antigo da raiz tenha sido extraviado e que ninguém jamais terá acesso ao mesmo. Se o caso for o roubo da chave da AC-Raiz, seu certificado deverá ser revogado.

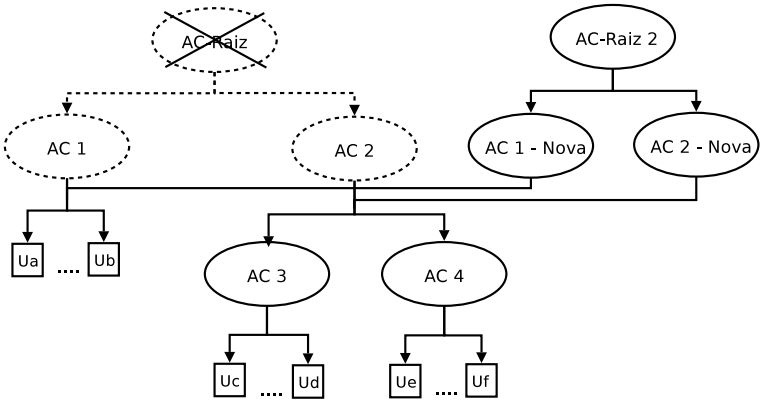

Figura 4.3: Processo de recriação de uma AC-Raiz com par de chaves corrompido, mantendo sua árvore de certificação antiga.

4.2.2 Atualização do certificado

Quando não houver a necessidade de substituição do par de chaves da AC, nem a alteração dos stributos do certificado, o processo de atualização é trivial. Um novo certificado auto-assindado é emitido, mantendo-se todas as informações do certificado antigo, com exceção do período de validade e do número serial. Assim, a árvore de certificação não será afetada pois o par de chaves do topo não terá sido alterado e os certificados já emitidos continuarão a reconhecê-lo, assim que o novo certificado seja aceito como confiável.

Se houver a necessidade de alteração dos atibutos do novo certificado, o processo torna-se um pouco mais delicado, uma vez que os campos *subject* e *issuer* não poderão ser alterados por serem usados na construção do caminho de certificação. Campos mais sucetíveis a mudança devem ser inseridos através da extensão *Subject Alternative Name*, como é o caso do endereço eletrônico, um campo que costumava compor o *subject*, mas segundo [HOU 02], novos certificados devem adicioná-lo por meio da extensão. Alguns certificados mantém o endereço de email compondo ambos os campos (*subject* e *Subject Alternative Name*) por questão de compatibilidade. Porém, campos do certificado que não são usados na construção do caminho de certificação, podem ser alterados conforme necessário.

A figura 4.4 mostra os procedimentos envolvidos na atualização do certificado de uma AC-Raiz. Um novo certificado é emitido, preservando-se o par de chaves da AC-Raiz, bem como seus campos *subject* e *issuer* (AC-Raiz nova validade). Uma vez que o mesmo par de chaves foi mantido, os certificados passarão a confiar também no novo certificado da AC.

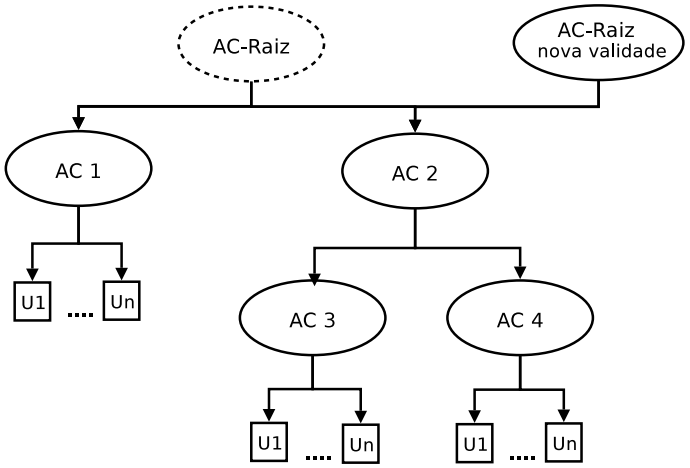

Figura 4.4: Processo de Renovação do certificado de uma AC-Raiz preservando-se o par de chaves antigo

Capítulo 5

Implementação

5.1 Simulação da Infra-estrutura de Chaves Públicas Brasileira

Para a simulação da ICP-Brasil foi criada uma ferramenta, batizada com o nome de JPKI. Essa ferramenta tem como finalidade principal a criação e gerenciamento de uma ICP, além de prover meios para os testes das propostas apresentadas em 4.2.1 e 4.2.2.

5.1.1 Funcionalidades

As principais funcionalidades implementadas na aplicação JPKI são:

- Criação de AC
- Emissão de Certificados
- Revogação de Certificados
- Povoamento da ICP
- Renovação de Certificados
- Subordinação de uma AC

A seguir, é feita uma demonstração de uso do programa, além de uma explicação mais detalhada de cada funcionalidade do mesmo.

Criação de AC

`jpki newca <caconfig> <certoutput>`

Permite a criação de autoridades certificadoras, raízes ou subordinadas, a partir de um arquivo de entrada em XML (`caconfig`), como o usado para a geração da AC-Raiz, cujo conteúdo é mostrado a seguir e exporta o certificado gerado para o caminho fornecido em `certoutput`.

A partir desse arquivo de entrada, o programa passa a gerenciar uma autoridade certificadora com o nome interno especificado pelo atributo `name`, presente na tag XML `<ca>`. Este nome deverá estar presente no campo issuer do arquivo de configuração de um certificado digital, mostrado a seguir.

Como o exemplo abaixo é de uma AC-Raiz, o mesmo campo presente em `name`, aparece também como `issuer`do certificado que será emitido na criação da AC, caracterizando assim um certificado auto-assinado.

```
<?xml version="1.0" encoding="UTF-8"?>
<ca name="ac-raiz">
<certificate>
 <issuer name="ac-raiz"/>
 <subject>
  <cn>Autoridade Certificadora Raiz Brasileira</cn>
  <st>DF</st>
  <l>Brasilia</l>
  <ou>Instituto Nacional de Tecnologia da Informacao - ITI</ou>
  <o>ICP-Brasil</o>
  <c>BR</c>
 </subject>
```

```
 <validity>10</validity> <!-- in years -->
 <extensions>
  <extension type="authKeyId">
   <type>basic</type> <!-- basic,nodir,noserial,complete -->
  </extension>
  <extension type="keyUsage">
   <keyCertSign/>
   <cRLSign/>
  </extension>
  <extension type="certificatePolicy">
   <oid>2.16.76.1.1.0</oid>
   <cps>http://acraiz.icpbrasil.gov.br/DPCacraiz.pdf</cps>
  </extension>
  <extension type="crlDistributionPoint">
   <uri>http://url.ufsc.br</uri>
  </extension>
 </extensions>
</certificate>
</ca>
```

Emissão de Certificados

```
jpki newcert <certconfig> <certoutput>
```

Emite certificados para entidades finais, a partir de um arquivo de XML contendo dados do certificado (`certconfig`) e exporta o certificado gerado para `certoutput`. O conteúdo do arquivo XML representando um certificado é mostrado a seguir:

```
<certificate>
 <issuer name="ac-serpro"/>
```

```
 <subject>
  <cn>Jose da Silva</cn>
  <o>UFSC</o>
  <ou>Laboratorio de Seguranca em Computacao</ou>
  <l>Florianopolis</l>
  <st>SC</st>
  <c>BR</c>
 </subject>
 <validity>1</validity>
 <extensions>
  <extension type="keyUsage">
   <digitalSignature/>
   <nonRepudiation/>
   <dataEncipherment/>
  <extension>
 </extensions>
</certificate>
```

Revogação de Certificados

`jpki revoke <certserial> <issuerlabel> <reasoncode>`

Revoga certificados a partir de seu serial (`certserial`), seu emissor (`issuerlabel`) e um terceiro parâmetro contendo o código referente à razão da revogação, de acordo com a extensão ”Reason Code” presente na especificação X.509 (`reasoncode`) [HOU 02].

Povoamento da ICP

`jpki simulation <namesfile> <numcerts> <outputpath>`

Emite `numcerts` certificados de entidades finais para cada autoridade certificadora criada previamente, colocando-os em `outputpath`. Para isso, usa uma

lista de nomes lida de um arquivo para povoar o campo subject de cada certificado `namesfile`.

Este método foi criado para automatizar a emissão de certificados para entidades finais e assim reproduzir uma infra-estrutura de chaves públicas de grande porte.

Renovação de Certificados

```
jpki renew <certfile>
```

Renova o certificado contido em `certfile`, caso o mesmo seja um certificado válido, sobrescrevendo o certificado antigo contido no mesmo arquivo. Caso o certificado antigo pertença a uma entidade final, o mesmo será revogado.

Essa funcionalidade serve para a aplicação dos testes necessários no caso da atualização do certificado da AC-Raiz 4.2.2.

Subordinação de uma AC

```
jpki copy <caname> <newcaname> <superca> <certoutput>
```

Esse método foi criado para a aplicação dos testes envolvendo a troca do par de chaves de uma autoridade certificadora, conforme exposto em 4.2.1. Com essa funcionalidade é possível copiar os dados pertencentes a uma determinada AC (`caname`), sobretudo o mesmo par de chaves, inserindo-a como subordinada de uma nova autoridade (`newcaname`). Assim, um novo certificado digital é gerado para a AC, assinado pela autoridade `superca`. O certificado da AC subordinada é armazenado em `certoutput`.

5.1.2 Tecnologias utilizadas

Java

Foi escolhida a linguagem de programação Java por prévia experiência com a ferramenta, por possuir uma vasta interface de programação (API) e documentação

disponíveis, por possuír ambientes de programação completos e de rápido desenvolvimento, além de ser orientada a objetos.

API criptográfica Bouncy Castle

A biblioteca Bouncy Castle [BOU] provê uma série de serviços criptográficos de forma muito organizada e fielmente baseada nos padrões e especificações de segurança. No caso deste trabalho, em específico, a biblioteca foi amplamente explorada em sua parte que implementa um provedor criptográfico e o padrão X.509.

Persistência dos Dados

Para mapeamento objeto-relacional, foi usada a ferramenta hibernate [HIB], que automatiza e facilita o acesso à base de dados, além de permitir a migração de um sistema de gerenciamento de banco de dados para outro de forma simples.

Como sistema de gerenciamento de banco de dados, foi utilizada a ferramenta HSQLDB [HSQ], por sua fácil operação e integração com hibernate e java.

5.2 Renovador Online de Certificados Digitais

Como alternativa para o problema da substituição dos certificados de toda uma cadeia, foi criada uma ferramenta para a renovação online de certificados digitais pertencentes a entidades finais.

Tal ferramenta deve atuar permitindo aos usuários credenciados à ICP a renovação de seus certificados digitais, sem a necessidade de submeter uma nova requisição para a AC. Dessa maneira, o próprio certificado digital antigo deveria ser submetido para a substituição por um novo, Cabendo à AC que o emitiu avaliar e validar ou não a nova emissão, possibilitando a obtenção do certificado instantaneamente. Após o envio do novo certificado, o certificado antigo da entidade é revogado, caso ainda esteja válido.

A Renovação do certificado pode ocorrer de duas maneiras diferentes:

- **Via SSL:** uma conexão segura é estabelecida entre o cliente e o servidor, com autenticação segura de ambas as partes, sendo que o próprio certificado usado para autenticar o cliente é o candidato a renovação. Sendo assim, para realizar esse tipo de renovação o usuário deve ter sua identidade digital (certificado e chave privada correspondente) configurada em seu navegador;

- **Via *upload* de arquivo:** Essa é uma forma alternativa de envio caso o navegador não tenha suporte ao uso de certificação digital ou caso o usuário não tenha sua identida configurada no mesmo. Através desse método, o certificado do usuário é selecionado no disco e enviado para o servidor, para que seja renovado. Para garantir a posse da chave privada, um email cifrado com a chave pública contida no certificado do usuário é enviada para um email informado pelo usuário no momento do *upload*. O email contém um desafio que deve ser respondido pelo usuário, para dar continuidade ao processo de renovação.

As figuras 5.1 a 5.7, em 5.2.2, ilustram o funcionamento do programa, conforme descrito.

5.2.1 Tecnologias utilizadas

PHP

Para a camada web da aplicação foi utilizada a linguagem PHP, por ser uma linguagem de desenvolvimento rápido de aplicações web. PHP conta ainda com uma comunidade muito ampla e, com isso, muita documentação disponível para auxiliar no desenvolvimento dos mais variáveis tipos de aplicações.

Zend Framework

Zend Framework é um conjunto de classes para PHP, usando a orientação a objetos que as novas versões de PHP proporcionam. Esse framework incentiva a programação em camadas e torna a organização do código muito simples, além de

tratar de questões como redirecionamento de páginas de maneira automática e transparente para o programador.

Servidor Web Apache

Foi usado o servidor web apache por ser uma ferramenta livre, gratuita, amplamente utilizada e por suportar o uso de autenticação segura SSL.

5.2.2 Ilustração do Funcionamento do Renovador de Certificados

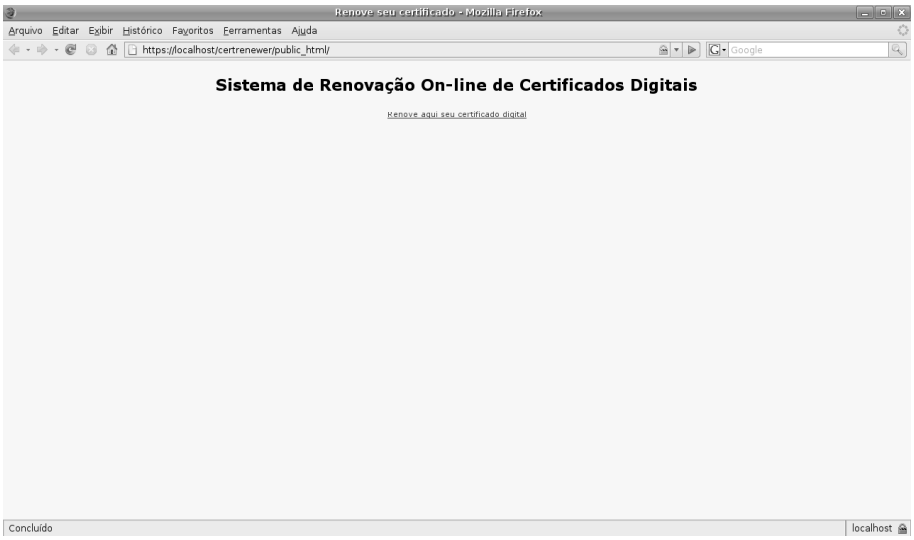

Figura 5.1: Tela inicial do renovador *on-line* de certificados digitais

Figura 5.2: Tela de escolha do método de renovação

Figura 5.3: Tela de download do certificado renovado

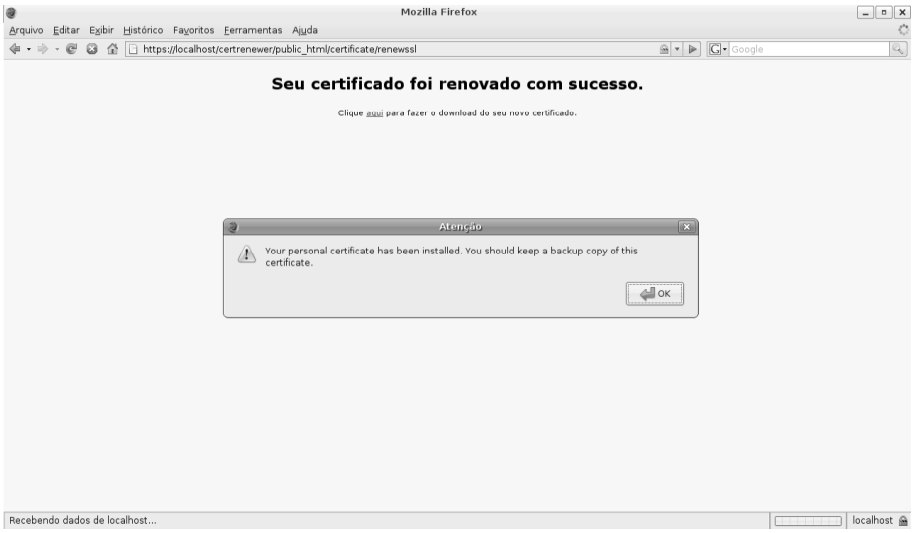

Figura 5.4: Tela ilustrando a importação do novo certificado

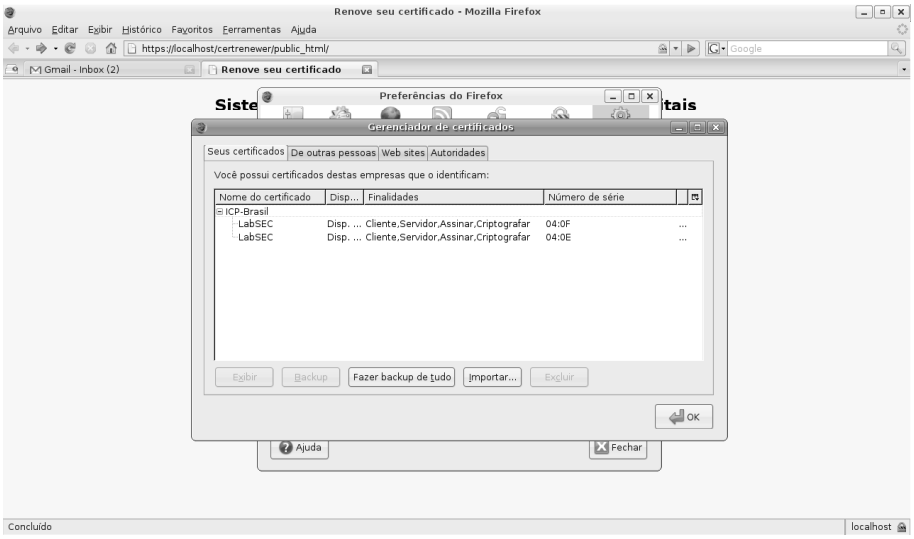

Figura 5.5: Tela mostrando o certificado renovado junto do certificado antigo, que agora foi revogado

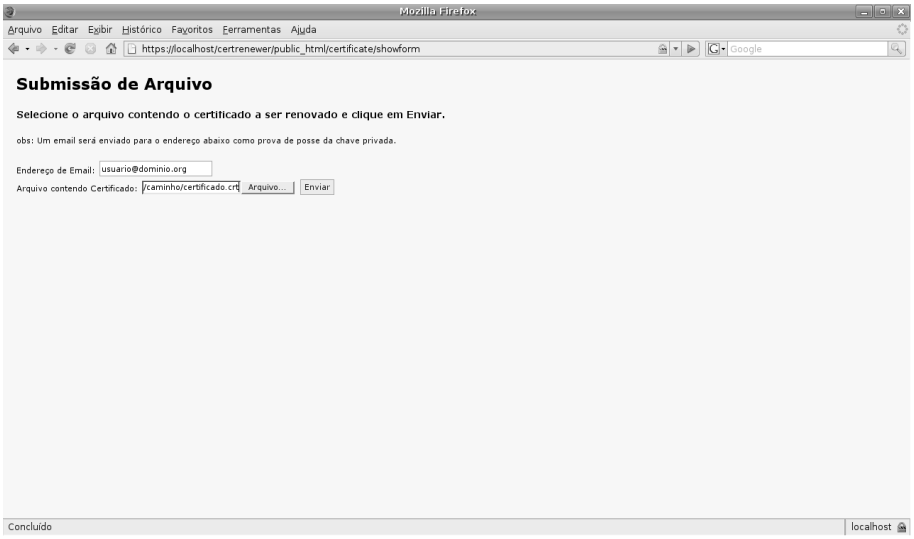

Figura 5.6: Tela mostrando método alternativo de renovação através do *upload* do arquivo contendo o certificado a ser renovado

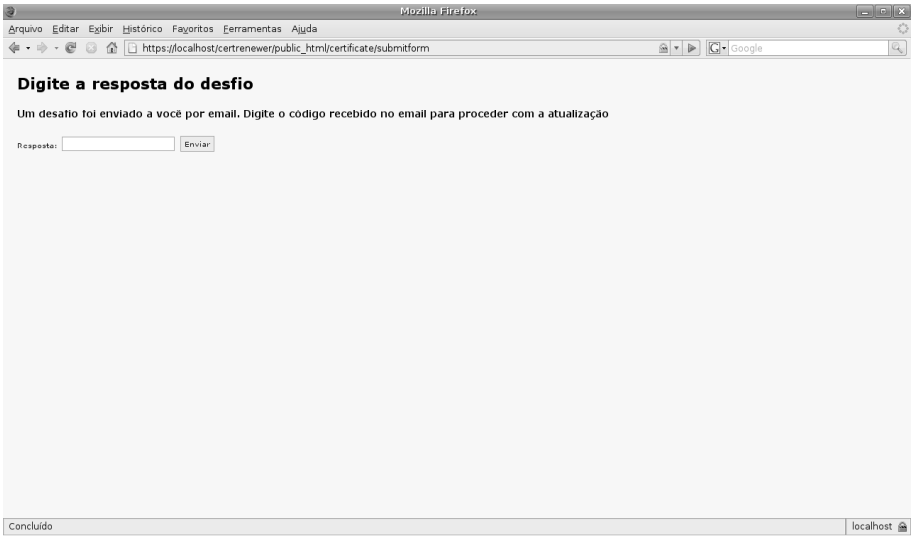

Figura 5.7: Tela ilustrando a resposta ao desafio enviado para o email informado.

Capítulo 6

Solução

Para testar as propostas apresentadas em 4.2.2 e 4.2.1, foi criada uma infra-estrutura semelhante à atual ICP-Brasil, conforme a figura mostrada em 3.5

Tal hierarquia foi então reproduzida, tendo a AC-Raiz brasileira como ponto máximo de confiança, que vamos chamar de nível 0. Logo abaixo, temos as ACs subordinadas diretamente à AC-Raiz, diremos que elas se encontram no nível 1. A seguir, subordinadas às ACs de nível 1, temos as demais autoridades, que serão chamadas de ACs de nível 2.

Após a reprodução da hierarquia de ACs, cerca de 1000 certificados foram emitidos para cada AC de nível 2, para representar os certificados de usuários finais.

Com a estrutura da ICP-Brasil pronta, os testes foram aplicados, como será mostrado em 6.1 e 6.2.

As ferramentas usadas para validar os caminhos de certificação foram:

- Mozilla Firefox;
- Microsoft Internet Explorer;
- Ferramenta OpenSSL.

A ferramenta OpenSSL não fornece informações detalhadas sobre a montagem e validação dos caminhos de certificação candidatos. A saída do programa é

praticamente booleana, ou seja, OK em caso de sucesso e ERRO em caso de falha, não mostrando o caminho escolhido. Além disso, segundo a própria documentação do OpenSSL, a verificação do caminho de certificação exige que cada campo da extensão *Authority Key Identifier* seja igual ao do certificado emissor para o sucesso da validação. Essa regra não está em conformidade com as normas de construção do caminho, em que o par "serial" e "identificador do emissor" deveriam servir apenas como indicação para a montagem do caminho, não como ponto crucial [IT 98, LOI 02]. Com essa ressalva, os seguintes resultados foram observados:

6.1 Atualização do certificado de uma AC

Com relação à proposta de atualização do certificado de uma AC, os seguintes procedimentos foram aplicados à estrutura acima descrita:

- Os caminhos de certificação iniciais foram estabelecidos;
- O certificado da autoridade certificadora raiz foi atualizado segundo a proposta descrita em 4.2.2;
- O novo certificado foi importado, mantendo-se o certificado antigo;
- Os caminhos de certificação foram novamente conferidos;
- O certificado antigo da AC-Raiz foi removido;
- Os caminhos de certificação foram novamente conferidos;

Além dos passos acima, vale ressaltar que os três tipos possíveis de *Authority Key Identifier* (AKID) foram testados. Conténdo apenas o identificador da chave da autoridade (básico), com o identificador da chave mais serial do certificado do emissor e ainda um terceiro caso que seria a forma completa, incluindo assim o nome do emissor ao anterior.

Dados os procedimentos acima, o comportamento esperado, segundo a literatura [IT 98, LOI 02], seria a preferência pelo certificado mais atual, salvo quando

o AKID estivesse completo, com o par emissor e serial. Nesse caso o caminho de certificação preferido deveria ser aquele que levasse ao certificado antido, onde tais valores correspondessem aos encontrados na extensão.

6.1.1 Resultados obtidos

Com a aplicação dos procedimentos acima citados, os seguintes comportamentos foram observados:

Internet Explorer

Com o AKID básico e ambos os certificados disponíveis (novo e antigo), o caminho validado foi o que levou ao novo certificado. Quando o AKID possuía também o serial do emissor, o comportamento foi o mesmo, ou seja, validando para o novo certificado. Com o uso do AKID completo, o caminho preferido foi o que levava ao certificado antigo, onde os campos serial e sujeito eram os mesmos contidos na extensão AKID. Após a remoção do certificado antigo como ponto de confiança, o caminho foi validado até o novo certificado, para todos os tipos de AKID.

Sendo assim, o navegador mostrou um comportamento que correspondeu perfeitamenente ao esperado segundo as referências citadas anteriormente, ou seja, o uso do serial e nome do emissor apenas como uma preferência entre dois caminhos candidatos.

Mozilla Firefox

Com o AKID básico e ambos os certificados disponíveis (novo e antigo), o caminho validado foi o que levou ao novo certificado. Quando o AKID possuía também o serial do emissor, o comportamento foi o mesmo, ou seja, validando para o novo certificado. Com o uso do AKID completo, o caminho preferido foi o que levava ao certificado antigo, onde o campo serial era o mesmo contido na extensão AKID. Após a remoção do certificado antigo como ponto de confiança, entretanto , o caminho não foi validado, em detrimento ao que ocorreu com o navegador Internet Explorer.

Dessa forma, o navegador Mozilla Firefox mostrou um comportamento não corresponde ao esperado segundo as referências citadas anteriormente, pois não usou os campos número serial e emissor apenas como uma forma de preferenciar um caminho a outro, mas sim como requisito para a validação do caminho de certificação.

OpenSSL

Conforme exposto no início da seção, a ferramenta OpenSSL mostrou algumas limitações, logo os resultados observados não foram muito detalhados. Com ambos os certificados presentes, o caminho de certificação foi validado com sucesso. Ao se remover o certificado antigo, o caminho foi validado somente com o AKID básico, ou seja, apenas com o identificador da chave do emissor.

Esse comportamento também não condiz com o esperado. Contudo, conforme exposto, a própria documentação do OpenSSL deixa claro que os campos número serial e nome do emissor são comparados na construção do caminho de certificação e são fatores determinantes no processo.

6.1.2 Resumo dos Resultados Obtidos

A tabela 6.1 mostra uma síntese dos comportamentos observados na atualização do certificado da AC-Raiz:

	Tipos de Authority Key Identifier (AKID)		
Ferramenta	**AKID básico**	**AKID com serial**	**AKID completo**
Internet Explorer	✓	✓	✓
Mozilla Firefox	✓	✓	×
OpenSSL	✓	×	×

Tabela 6.1: Resultados obtidos no procedimento de atualização do certificado da AC-Raiz.

6.2 Susbstituição do par de chaves da AC-Raiz

Para testar a substituição do par de chaves da AC-Raiz, foram seguidos os passos abaixo:

- Os caminhos de certificação iniciais foram estabelecidos;
- Uma nova AC-Raiz foi criada, com um novo par de chaves e um novo certificado auto-assinado;
- As autoridades certificadoras de nível 1 foram subordinadas à nova AC-Raiz;
- Os novos certificados gerados foram importados mantendo-se os antigos;
- Os caminhos de certificação foram conferidos;
- Os certificados antigos das ACs de nível 0 e 1 foram removidos;
- Os caminhos de certificação foram novamente conferidos;

Além dos passos acima, vale ressaltar que os três tipos possíveis de *Authority Key Identifier* (AKID) foram testados. Conténdo apenas o identificador da chave da autoridade (básico), com o identificador da chave mais serial do certificado do emissor e ainda um terceiro caso que seria a forma completa, incluindo assim o nome do emissor ao anterior.

6.2.1 Resultados obtidos

O procedimento de substituição do par de chaves da AC-Raiz foi testado de duas maneiras distintas, no primeiro, cujos resultados estão sintetizados em 6.2, foi reproduzido o certificado da nova AC-Raiz semelhante ao certificado original. No segundo teste, foi alterado o campo *subject* do novo certificado, para analisar o impacto que haveria com os certificados que utilizávam a extensão *Authority Key Identifier* completa (com o *subject* da AC-Raiz antiga). Uma tabela com os resultados do segundo teste é mostrada em 6.3.

Internet Explorer

O Internet Explorer apresenou o mesmo comportamento nos dois testes, mesmo quando o *subject* foi alterado, fazendo com que certificados em níveis mais baixos da hierarquia apontassem para um *issuer* não mais existente. Sendo assim, o resultado encontrado foi semelhante ao observado na atualização do certificado da AC-Raiz, sendo detalhado abaixo.

Com o AKID básico e ambos os certificados disponíveis (novos e antigos), o caminho validado foi o que levou aos novos certificados. Quando o AKID possuía também o serial do emissor, o comportamento foi o mesmo, ou seja, validando para os novos certificados. Com o uso do AKID completo, o caminho preferido foi o que levava aos certificados antigos, onde os campos serial e sujeito eram os mesmos contidos na extensão AKID. Após a remoção dos certificados antigos como ponto de confiança, o caminho foi validado até os novos certificados, para todos os tipos de AKID.

Sendo assim, o navegador mostrou um comportamento que correspondeu perfeitamenente ao esperado segundo as referências citadas anteriormente, ou seja, o uso do serial e nome do emissor apenas como uma preferência entre dois caminhos candidatos.

Mozilla Firefox

O Navegador Mozilla Firefox não permitiu a importação de certificados com chaves públicas diferentes que possuíssem o mesmo par (serial, sujeito). Dessa forma não foi possível comparar a validação do caminho de certificação com ambos os certificados, novos e antigos, quando os novos certificados eram semelhantes aos antigos, ou seja, quando os certificados novos e antigos só diferiam em suas chaves públicas e datas de validade, além das extensões relacionadas à chave.

Quando o sujeito do certificado da AC-Raiz e a ordem de emissão dos certificados das ACs de nível 1 foram alterados (mudando seus seriais em relação aos antigos), os certificados puderam ser importados simultaneamente e os seguintes resultados foram observados: com o AKID básico e ambos os certificados disponíveis

(novos e antigos), o caminho validado foi o que levou aos novos certificados. Quando o AKID possuía também o serial do emissor, o comportamento foi o mesmo, ou seja, validando para os novos certificados. Com o uso do AKID completo, o caminho preferido foi o que levava aos certificados antigos, onde os campos serial e sujeito eram o mesmos contidos na extensão AKID. Após a remoção dos certificados antigos como ponto de confiança, contudo, o caminho não foi validado, em detrimento ao que ocorreu com o navegador Internet Explorer.

Dessa forma, o navegador Mozilla Firefox mostrou um comportamento não corresponde ao esperado segundo as referências citadas anteriormente, pois não usou os campos número serial e emissor apenas como forma de preferenciar um caminho a outro, mas sim como requisito para a validação do caminho de certificação, provocando a reprovação de um caminho válido.

OpenSSL

Conforme exposto no início da seção, a ferramenta OpenSSL mostrou algumas limitações, logo os resultados observados não foram muito detalhados. Com ambos os certificados presentes, novos e antigos, o caminho de certificação foi validado com sucesso. Ao se remover os certificados antigos, o caminho foi validado somente com o AKID básico, ou seja, apenas com o identificador da chave do emissor.

Esse comportamento também não condiz com o esperado. Contudo, conforme exposto, a própria documentação do OpenSSL deixa claro que os campos número serial e nome do emissor são comparados na construção do caminho de certificação e são fatores determinantes no processo.

6.2.2 Resumo dos Resultados Obtidos

As tabelas a seguir mostram um resumo do comportamento observado no processo de substituição do par de chaves de uma AC-Raiz. Em 6.2 a substituição do par de chaves mantendo os dados dos certificados antigos nos novos certificados. Na tabela 6.3, a mesma substituição, porém com a alteração do campo sujeito do certifi-

cado da AC-Raiz.

O símbolo ⊗ faz referência ao comportamento do OpenSSL, que trata os campos serial e emissor do AKID obrigatórios. Logo, quando esses campos eram iguais, a validação ocorreu normalmente. Quando a ordem de emissão dos certificados de nível 1 não foi respeitada, gerando certificados com seriais diferentes dos respectivos antigos, o caminho foi reprovado.

	Tipos de Authority Key Identifier (AKID)		
Ferramenta	**AKID básico**	**AKID com serial**	**AKID completo**
Internet Explorer	✓	✓	✓
Mozilla Firefox	✓	✓	✓
OpenSSL	✓	⊗	⊗

Tabela 6.2: Tabela contendo os resultados obtidos na substituição do par de chaves da AC-Raiz com certificado novo semelhante ao antigo

	Tipos de Authority Key Identifier (AKID)		
Ferramenta	**AKID básico**	**AKID com serial**	**AKID completo**
Internet Explorer	✓	✓	✓
Mozilla Firefox	✓	✓	×
OpenSSL	✓	×	×

Tabela 6.3: Tabela contendo os resultados obtidos na substituição do par de chaves da AC-Raiz com certificado novo diferente do antigo

Capítulo 7

Considerações Finais

De acordo com os testes aplicados, as alternativas propostas para a troca do par de chaves de uma AC-Raiz, bem como a atualização de seu certificado, se mostraram úteis e aplicáveis, cada uma em seu devido contexto.

No caso da atualização do certificado, o método se mostrou de fácil aplicação e de baixo custo, uma vez que envolve a reemissão de apenas um certificado, mantendo a hiearquia intacta, agregando assim mais confiabilidade a uma ICP a longo prazo.

No caso da recriação da autoridade certificadora raiz, pela geração de um novo par de chaves e certificado, o método proposto mostrou ser excelente em relação aos pré-existentes, uma vez que reduziu a sensivelmente a quantidade de certificados envolvidos no processo, tratando o caso mais crítico no gerenciamento de uma ICP, quando o par de chaves de uma AC-Raiz foi comprometido.

Apesar dos problemas encontrados na montagem do caminho de certificação em algumas das situações, acredita-se que tais particularidades são inerentes à ferramenta utilizada, não à técnica em si, uma vez que ocorreram em situações específicas de uso, sendo que em outras ferramentas o resultado foi positivo.

Ainda para o caso da criação de uma nova estrutura baseada na anterior, foi proposto e implementado um terceiro método que automatiza a renovação de certificados, dispensando o uso tradicional de requisições, facilitando o processo de migração, representando também uma queda de custo da transição.

Referências

[ADA 00] ADAMS, C. et al. Which pki (public key infrastructure) is the right one? (panel session). In: CCS '00: PROCEEDINGS OF THE 7TH ACM CONFERENCE ON COMPUTER AND COMMUNICATIONS SECURITY, 2000. **Proceedings...** New York, NY, USA: ACM Press, 2000. p.98–101.

[ADA 02] ADAMS, C.; LLOYD, S. **Understanding PKI: Concepts, Standards, and Deployment Considerations**. Addison Wesley, 2002.

[ADA 05] ADAMS, C. et al. **Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)**. RFC 4210 (Proposed Standard).

[BOU] BOUNCYCASTLE. **Bouncy Castle Crypto APIs**. Disponível em <http://www.bouncycastle.org/java.html>. Acesso em: 06.2007.

[CAR 07] CARLOS, M. C. Best practices for long-term key management in a public key infrastructure. **ACM?**, [S.l.], v.?, p.7, 2007.

[CAR to] CARLOS, M. C. **Substituição Dinâmica de ACs e Chaves Privadas**. Universidade Federal de Santa Catarina, 2007 (em andamento). Dissertação de Mestrado.

[FRE 98] FREEMAN, T. Certificate trust lists: What are they? why are they useful? presentation at the NIST PKI Working Group meeting, November, 1998.

[HIB] HIBERNATE. **Hibernate Framework**. Disponível em <http://www.hibernate.org/>. Acesso em: 06.2007.

[HOU 01] HOUSLEY, R.; POLK, T. **Planning for PKI: Best Practices Guide for Deploying Public Key Infrastructure**. New York, NY, USA: John Wiley & Sons, Inc., 2001.

[HOU 02] HOUSLEY, R. et al. **Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile**. RFC 3280 (Proposed Standard). Updated by RFCs 4325, 4630.

[HSQ] HSQLDB. **Sistema de Gerenciamento de Banco de Dados HSQLDB**. Disponível em <http://hsqldb.org/>. Acesso em: 06.2007.

[ICP a] ICPBRASIL. **Estrutura da ICP-Brasil**. Disponível em <http://www.iti.gov.br/twiki/pub/Certificacao/EstruturaIcp/Estrutura_da_ICP_.pdf>. Acesso em: 25 de fevereiro de 2006.

[ICP b] ICPBRASIL. **ICP Brasil: Infra-estrutura de Chaves Públicas Brasileira**. Disponível em <http://www.icpbrasil.gov.br/>. Acesso em: 05 de junho de 2007.

[IT 98] ITU-T. Information technology - open systems interconnection - the directory: Authentication framework. International Telecomunication Union, 1998. Relatório técnico.

[JAN] JANUáRIO, L. **Número de certificados digitais deve chegar a 1 milhão em 2006, diz ITI**. Disponível em <http://www.serpro.gov.br/noticiasSERPRO/20060509_01>. Acesso em: 06.2007.

[LOI 02] LOID, S. Understanding path construction. PKI Forum, 2002. Relatório técnico.

[MAR 05] MARTINA, J. E. **Projeto de um Provedor de Serviços Criptográficos Embarcado para Infra-estrutura de Chaves Públicas e suas Aplicações**. Universidade Federal de Santa Catarina, 2005. Dissertação de Mestrado.

[MEN 96] MENEZES, A. J.; VANSTONE, S. A.; OORSCHOT, P. C. V. **Handbook of Applied Cryptography**. Boca Raton, FL, USA: CRC Press, Inc., 1996.

[NIS 77] NIST. Data encryption standart. National Bureal of Standards, 1977. Federal information processing standards publication 46.

[NIS 92] NIST, C. The digital signature standard. **Commun. ACM**, New York, NY, USA, v.35, n.7, p.36–40, 1992.

[NIS 94] NIST. Digital signature standard. National Institute of Standards and Technology - Department of Commerce, May, 1994. Federal information processing standards publication 186.

[NIS 01] NIST. Advanced encryption standard (aes). National Institute of Standards and Technology, 2001. Technical report.

[NYS 00] NYSTROM, M.; KALISKI, B. **PKCS #10: Certification Request Syntax Specification Version 1.7**. RFC 2986 (Informational).

[ONL] ONLINE, F. **Comércio eletrônico cresce 76% e movimenta R$ 4,4 bi em 2006**. Disponível em <http://www1.folha.uol.com.br/folha/dinheiro/ult91u114374.shtml>. Acesso em: 05 de junho de 2007. publicado em 08 de fevereiro de 2007.

[PFL 89] PFLEEGER, C. P. **Security in Computing**. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.

[SAL 96] SALOMAA, A. **Public Key Cryptography**. Springer, 1996.

[SCH 93] SCHNEIER, B. **Applied Cryptography: Protocols, Algorithms, and Source Code in C**. New York, NY, USA: John Wiley & Sons, Inc., 1993.

[SZY 05] SZYDLO, M.; YIN, Y. L. Collision-resistant usage of md5 and sha-1 via message processing. RSA Laboratories,, 2005. Relatório técnico.

[TEC 93] TECHNICAL, R. L. Pckcs #7 - cryptographic message syntax standard. RSA Data Security, November, 1993. Technical specification.

[TEC 02] TECHNICAL, R. L. Pkcs #1: Rsa cryptography standard v2.1. RSA Data Security, 2002. Relatório técnico.

Apêndice A

Código Fonte JPKI

```
package br.ufsc.jpki.ca;

import java.io.IOException;
import java.math.BigInteger;
import java.security.InvalidKeyException;
import java.security.KeyStore;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.SignatureException;
import java.security.cert.CertificateException;
import java.security.cert.CertificateParsingException;
import java.security.cert.X509CRL;
import java.security.cert.X509Certificate;
import java.util.Calendar;
import java.util.Collection;
import java.util.Date;
import java.util.Iterator;
import java.util.List;
import java.util.Set;
import java.util.Vector;

import javax.security.auth.x500.X500Principal;

import org.bouncycastle.asn1.ASN1Encodable;
import org.bouncycastle.asn1.ASN1EncodableVector;
import org.bouncycastle.asn1.ASN1InputStream;
```

```
import org.bouncycastle.asn1.ASN1Sequence;
import org.bouncycastle.asn1.DEREncodable;
import org.bouncycastle.asn1.DERInteger;
import org.bouncycastle.asn1.DEROctetString;
import org.bouncycastle.asn1.DERSequence;
import org.bouncycastle.asn1.DERTaggedObject;
import org.bouncycastle.asn1.pkcs.CertificationRequest;
import org.bouncycastle.asn1.x509.AlgorithmIdentifier;
import org.bouncycastle.asn1.x509.AuthorityKeyIdentifier;
import org.bouncycastle.asn1.x509.CRLNumber;
import org.bouncycastle.asn1.x509.GeneralName;
import org.bouncycastle.asn1.x509.GeneralNames;
import org.bouncycastle.asn1.x509.KeyUsage;
import org.bouncycastle.asn1.x509.SubjectKeyIdentifier;
import org.bouncycastle.asn1.x509.SubjectPublicKeyInfo;
import org.bouncycastle.asn1.x509.X509Extensions;
import org.bouncycastle.asn1.x509.X509Name;
import org.bouncycastle.crypto.Digest;
import org.bouncycastle.crypto.digests.SHA1Digest;
import org.bouncycastle.jce.PrincipalUtil;
import org.bouncycastle.jce.provider.X509CertificateObject;
import org.bouncycastle.x509.X509V2CRLGenerator;
import org.bouncycastle.x509.X509V3CertificateGenerator;
import org.bouncycastle.x509.extension.AuthorityKeyIdentifierStructure;
import org.bouncycastle.x509.extension.SubjectKeyIdentifierStructure;
import org.bouncycastle.x509.extension.X509ExtensionUtil;

import sun.security.x509.AlgorithmId;

import br.ufsc.jpki.container.CertParameters;
import br.ufsc.jpki.exception.CertificateNotFoundException;
import br.ufsc.jpki.exception.CertificationAuthorityBuildingException;
import br.ufsc.jpki.exception.CertificationAuthorityException;
import br.ufsc.jpki.exception.DataLoadingException;
import br.ufsc.jpki.exception.DataWritingException;
import br.ufsc.jpki.persistence.HCertificateDAO;
import br.ufsc.jpki.persistence.HCertificationAuthorityDAO;
import br.ufsc.jpki.persistence.PersistentCertificationAuthority;
import br.ufsc.jpki.persistence.RevokedCertificate;
import br.ufsc.jpki.persistence.dao.CertificateDAO;
import br.ufsc.jpki.persistence.dao.CertificationAuthorityDAO;

public class CertificationAuthority {
```

```
private String label;

private int level;

private BigInteger lastCertSerial;

private BigInteger lastCrlSerial;

private X509Certificate certificate;

private PrivateKey privateKey;

private AuthKeyIdType authKeyIdType;

public enum AuthKeyIdType {

BASIC, NODIR, NOSERIAL, COMPLETE;

public int toInt() {
switch (this) {
case NODIR:
return 1;
case NOSERIAL:
return 2;
case COMPLETE:
return 3;
default:
return 0;
}
}

public static AuthKeyIdType fromInt(int tval) {
switch (tval) {
case 1:
return NODIR;
case 2:
return NOSERIAL;
case 3:
return COMPLETE;
default:
return BASIC;
}
}
```

```
};

public CertificationAuthority(String label, int level,
X509Certificate cert, PrivateKey prk, AuthKeyIdType t) {
this.level = level;
this.authKeyIdType = t;
this.label = label;
this.lastCertSerial = new BigInteger("1");
this.lastCrlSerial = new BigInteger("0");
this.certificate = cert;
this.privateKey = prk;
}

public CertificationAuthority(CertificationAuthority aCA) {
this.level = aCA.level;
this.authKeyIdType = aCA.authKeyIdType;
this.label = aCA.label;
this.lastCertSerial = aCA.lastCertSerial;
this.lastCrlSerial = aCA.lastCrlSerial;
this.certificate = aCA.certificate;
this.privateKey = aCA.privateKey;
}

public X509Certificate getCertificate() {
return certificate;
}

public void setCertificate(X509Certificate certificate) {
this.certificate = certificate;
}

public BigInteger getLastCertSerial() {
return this.lastCertSerial;
}

public BigInteger getNextCertSerial() {
this.lastCertSerial = this.lastCertSerial.add(new BigInteger("1"));
return this.lastCertSerial;
}

public BigInteger getNextCrlSerial() {
this.lastCrlSerial = this.lastCrlSerial.add(new BigInteger("1"));
return this.lastCrlSerial;
}
```

```
public PrivateKey getPrivateKey() {
return privateKey;
}

public String getLabel() {
return label;
}

public X509Certificate renewCertificate(X509Certificate oldCert,
Calendar notAfter, Calendar notBefore) throws CertificateException,
InvalidKeyException, SignatureException {

X509V3CertificateGenerator certGenerator = new X509V3CertificateGenerator();
certGenerator.setIssuerDN(this.certificate.getSubjectX500Principal());
certGenerator.setSubjectDN(oldCert.getSubjectX500Principal());
certGenerator.setPublicKey(oldCert.getPublicKey());
certGenerator.setSignatureAlgorithm(oldCert.getSigAlgName());
certGenerator.setNotAfter(notAfter.getTime());
certGenerator.setNotBefore(notBefore.getTime());
certGenerator.setSerialNumber(this.getNextCertSerial());
Set<String> cExts = oldCert.getCriticalExtensionOIDs();
Iterator<String> iter = cExts.iterator();
try {
while (iter.hasNext()) {
String extoid = (String) iter.next();
certGenerator.addExtension(extoid, true, X509ExtensionUtil
.fromExtensionValue(oldCert.getExtensionValue(extoid)));
}
Set<String> ncExts = oldCert.getNonCriticalExtensionOIDs();
iter = ncExts.iterator();
while (iter.hasNext()) {
String extoid = (String) iter.next();
certGenerator.addExtension(extoid, false, X509ExtensionUtil
.fromExtensionValue(oldCert.getExtensionValue(extoid)));
}
} catch (IOException e) {
throw new CertificateException(
"Error adding certificate extensions: "
+ e.getLocalizedMessage(), e);
}
return certGenerator.generateX509Certificate(this.privateKey);
}
```

```
public X509Certificate renewCertificate(X509Certificate oldCert,
Calendar notAfter) throws CertificateException,
InvalidKeyException, SignatureException {

return this.renewCertificate(oldCert, notAfter, Calendar.getInstance());
}

public X509Certificate issueCertificate(X509Certificate oldCert)
throws CertificateException, InvalidKeyException,
SignatureException {

X509V3CertificateGenerator certGenerator = new X509V3CertificateGenerator();
certGenerator.setIssuerDN(this.certificate.getSubjectX500Principal());
certGenerator.setSubjectDN(oldCert.getSubjectX500Principal());
certGenerator.setPublicKey(oldCert.getPublicKey());
certGenerator.setSignatureAlgorithm(oldCert.getSigAlgName());
certGenerator.setNotAfter(oldCert.getNotAfter());
certGenerator.setNotBefore(oldCert.getNotBefore());
certGenerator.setSerialNumber(this.getNextCertSerial());
SubjectKeyIdentifierStructure subjectKeyId = new SubjectKeyIdentifierStructure(
oldCert.getPublicKey());
certGenerator.addExtension(X509Extensions.SubjectKeyIdentifier, false,
subjectKeyId);
certGenerator.addExtension(X509Extensions.AuthorityKeyIdentifier,
false, this.buildAuthorityKeyIdentifier());
try {
if (oldCert.getExtensionValue(X509Extensions.KeyUsage.getId()) != null) {
certGenerator.addExtension(X509Extensions.KeyUsage, true,
X509ExtensionUtil.fromExtensionValue(oldCert
.getExtensionValue(X509Extensions.KeyUsage
.getId())));
}
if (oldCert.getExtensionValue(X509Extensions.BasicConstraints
.getId()) != null) {
certGenerator
.addExtension(
X509Extensions.BasicConstraints,
true,
X509ExtensionUtil
.fromExtensionValue(oldCert
.getExtensionValue(X509Extensions.BasicConstraints
.getId())));
}
if (oldCert.getExtensionValue(X509Extensions.CertificatePolicies
```

```
.getId()) != null) {
certGenerator
.addExtension(
X509Extensions.CertificatePolicies,
false,
X509ExtensionUtil
.fromExtensionValue(oldCert
.getExtensionValue(X509Extensions.CertificatePolicies
.getId())));
}
if (oldCert.getExtensionValue(X509Extensions.CRLDistributionPoints
.getId()) != null) {
certGenerator
.addExtension(
X509Extensions.CRLDistributionPoints,
false,
X509ExtensionUtil
.fromExtensionValue(oldCert
.getExtensionValue(X509Extensions.CRLDistributionPoints
.getId())));
}
} catch (Exception e) {
throw new CertificateException(
"Unable to add extensions from old certificate");
}
return certGenerator.generateX509Certificate(this.privateKey);
}

public X509Certificate issueCertificate(CertParameters cp, PublicKey pubkey)
throws InvalidKeyException, SecurityException, SignatureException,
NoSuchAlgorithmException, CertificateException {

X509V3CertificateGenerator certGenerator = new X509V3CertificateGenerator();
certGenerator.setIssuerDN(this.certificate.getSubjectX500Principal());
certGenerator.setSubjectDN(new X500Principal(cp.getSubject()));
certGenerator.setPublicKey(pubkey);
certGenerator.setSignatureAlgorithm(this.getCertificate()
.getSigAlgName());
certGenerator.setNotAfter(cp.getNotAfter().getTime());
certGenerator.setNotBefore(cp.getNotBefore().getTime());
certGenerator.setSerialNumber(this.getNextCertSerial());
SubjectKeyIdentifierStructure subjectKeyId = new SubjectKeyIdentifierStructure(
pubkey);
certGenerator.addExtension(X509Extensions.SubjectKeyIdentifier, false,
```

```
subjectKeyId);
certGenerator.addExtension(X509Extensions.AuthorityKeyIdentifier,
false, this.buildAuthorityKeyIdentifier());
certGenerator.addExtension(X509Extensions.KeyUsage, true, this
.buildKeyUsageExtension(cp.getKeyUsages()));
return certGenerator.generateX509Certificate(this.privateKey);
}

public void revokeCertificate(BigInteger certSerial, int reasonCode)
throws CertificateException {
try {
CertificateDAO cdao = new HCertificateDAO();
cdao.storeRevokedCertificate(certSerial, reasonCode, this
.getLabel());
} catch (Exception e) {
throw new CertificateException("Unable to revoke certificate:"
+ e.getMessage(), e);
}
}

public X509CRL issueCRL(List<RevokedCertificate> rcs, int validity)
throws SecurityException, SignatureException, InvalidKeyException,
CertificateException {

X509V2CRLGenerator crlGen = new X509V2CRLGenerator();
crlGen.setSignatureAlgorithm("SHA1withRSA");
Calendar thisUpdate = Calendar.getInstance();
Calendar nextUpdate = (Calendar) thisUpdate.clone();
nextUpdate.add(Calendar.DAY_OF_MONTH, validity);
crlGen.setThisUpdate(thisUpdate.getTime());
crlGen.setNextUpdate(nextUpdate.getTime());
crlGen.setIssuerDN(this.certificate.getIssuerX500Principal());
RevokedCertificate rc = null;
Iterator<RevokedCertificate> iter = rcs.iterator();
while (iter.hasNext()) {
rc = iter.next();
crlGen.addCRLEntry(rc.getSerial(), rc.getRevocationDate(), rc
.getReasonCode());
}
crlGen.addExtension(X509Extensions.AuthorityKeyIdentifier, false, this
.buildAuthorityKeyIdentifier());
crlGen.addExtension(X509Extensions.CRLNumber, false, new CRLNumber(this
.getNextCrlSerial()));
return crlGen.generateX509CRL(this.privateKey);
```

```
}

public BigInteger getLastCrlSerial() {
return lastCrlSerial;
}

public void setLastCrlSerial(BigInteger lastCrlSerial) {
this.lastCrlSerial = lastCrlSerial;
}

public void setPrivateKey(PrivateKey prk) {
this.privateKey = prk;
}

public void setLabel(String label) {
this.label = label;
}

public void setLastCertSerial(BigInteger lastCertSerial) {
this.lastCertSerial = lastCertSerial;
}

private KeyUsage buildKeyUsageExtension(
Vector<CertParameters.KeyPurpose> keyUsages) {
int keyUsageMask = 0;
if (keyUsages.contains(CertParameters.KeyPurpose.cRLSign)) {
keyUsageMask |= KeyUsage.cRLSign;
}
if (keyUsages.contains(CertParameters.KeyPurpose.dataEncipherment)) {
keyUsageMask |= KeyUsage.dataEncipherment;
}
if (keyUsages.contains(CertParameters.KeyPurpose.decipherOnly)) {
keyUsageMask |= KeyUsage.decipherOnly;
}
if (keyUsages.contains(CertParameters.KeyPurpose.digitalSignature)) {
keyUsageMask |= KeyUsage.digitalSignature;
}
if (keyUsages.contains(CertParameters.KeyPurpose.encipherOnly)) {
keyUsageMask |= KeyUsage.encipherOnly;
}
if (keyUsages.contains(CertParameters.KeyPurpose.keyAgreement)) {
keyUsageMask |= KeyUsage.keyAgreement;
}
if (keyUsages.contains(CertParameters.KeyPurpose.keyCertSign)) {
```

```
keyUsageMask |= KeyUsage.keyCertSign;
}
if (keyUsages.contains(CertParameters.KeyPurpose.keyEncipherment)) {
keyUsageMask |= KeyUsage.keyEncipherment;
}
if (keyUsages.contains(CertParameters.KeyPurpose.nonRepudiation)) {
keyUsageMask |= KeyUsage.nonRepudiation;
}
return new KeyUsage(keyUsageMask);
}

public boolean equals(Object obj) {
boolean equals = false;
if (obj instanceof CertificationAuthority) {
CertificationAuthority ca = (CertificationAuthority) obj;
equals = (this.certificate.equals(ca.certificate)
&& this.label.equals(ca.label)
&& this.lastCertSerial.equals(ca.lastCertSerial)
&& this.lastCrlSerial.equals(ca.lastCrlSerial) && this.privateKey
.equals(ca.privateKey));
}
return equals;
}

private AuthorityKeyIdentifier buildAuthorityKeyIdentifier()
throws InvalidKeyException, CertificateException {
// if (this.authKeyIdType == AuthKeyIdType.COMPLETE) {
// return new AuthorityKeyIdentifierStructure(this.certificate);
// }
// else {
DEREncodable keyidentifier = new SubjectKeyIdentifierStructure(
this.certificate.getPublicKey());
ASN1EncodableVector v = new ASN1EncodableVector();
v.add(new DERTaggedObject(false, 0, keyidentifier));
if (authKeyIdType != AuthKeyIdType.BASIC) {
if (this.authKeyIdType != AuthKeyIdType.NODIR) {
GeneralNames gns = new GeneralNames(new GeneralName(
PrincipalUtil.getIssuerX509Principal(this.certificate)));
v.add(new DERTaggedObject(false, 1, gns));
}
if (this.authKeyIdType != AuthKeyIdType.NOSERIAL) {
v.add(new DERTaggedObject(false, 2, new DERInteger(
this.certificate.getSerialNumber())));
}
```

```
}
return new AuthorityKeyIdentifier(new DERSequence(v));
// }
}

public AuthKeyIdType getAuthKeyIdType() {
return authKeyIdType;
}

public void setAuthKeyIdType(AuthKeyIdType authKeyIdType) {
this.authKeyIdType = authKeyIdType;
}

public CertificationAuthority addSubordinate(String newLabel,
CertificationAuthority subCA)
throws CertificationAuthorityException {
try {
X509Certificate newCert = this.issueCertificate(subCA
.getCertificate());
CertificationAuthority ca = new CertificationAuthority(newLabel,
this.level + 1, newCert, subCA.getPrivateKey(), subCA
.getAuthKeyIdType());
return ca;
} catch (Exception e) {
throw new CertificationAuthorityException(
"Unable to create subordinate certification authority: "
+ e.getLocalizedMessage(), e);
}
}

public int getLevel() {
return level;
}

public void setLevel(int level) {
this.level = level;
}

}
package br.ufsc.jpki.ca;

import java.math.BigInteger;
import java.security.InvalidKeyException;
import java.security.KeyPair;
```

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.cert.CertificateException;
import java.security.cert.CertificateParsingException;
import java.security.cert.X509Certificate;
import java.util.Calendar;
import java.util.Vector;

import javax.security.auth.x500.X500Principal;

import org.bouncycastle.asn1.ASN1Encodable;
import org.bouncycastle.asn1.ASN1EncodableVector;
import org.bouncycastle.asn1.ASN1InputStream;
import org.bouncycastle.asn1.ASN1Sequence;
import org.bouncycastle.asn1.DEREncodable;
import org.bouncycastle.asn1.DERInteger;
import org.bouncycastle.asn1.DERObjectIdentifier;
import org.bouncycastle.asn1.DEROctetString;
import org.bouncycastle.asn1.DERSequence;
import org.bouncycastle.asn1.DERTaggedObject;
import org.bouncycastle.asn1.pkcs.CertificationRequest;
import org.bouncycastle.asn1.x509.AuthorityKeyIdentifier;
import org.bouncycastle.asn1.x509.BasicConstraints;
import org.bouncycastle.asn1.x509.CRLDistPoint;
import org.bouncycastle.asn1.x509.CertPolicyId;
import org.bouncycastle.asn1.x509.DistributionPoint;
import org.bouncycastle.asn1.x509.DistributionPointName;
import org.bouncycastle.asn1.x509.GeneralName;
import org.bouncycastle.asn1.x509.GeneralNames;
import org.bouncycastle.asn1.x509.KeyUsage;
import org.bouncycastle.asn1.x509.PolicyInformation;
import org.bouncycastle.asn1.x509.PolicyQualifierId;
import org.bouncycastle.asn1.x509.PolicyQualifierInfo;
import org.bouncycastle.asn1.x509.SubjectKeyIdentifier;
import org.bouncycastle.asn1.x509.SubjectPublicKeyInfo;
import org.bouncycastle.asn1.x509.X509Extensions;
import org.bouncycastle.asn1.x509.X509Name;
import org.bouncycastle.crypto.Digest;
import org.bouncycastle.crypto.digests.SHA1Digest;
import org.bouncycastle.jce.PrincipalUtil;
import org.bouncycastle.x509.X509V3CertificateGenerator;
import org.bouncycastle.x509.extension.AuthorityKeyIdentifierStructure;
```

```
import org.bouncycastle.x509.extension.SubjectKeyIdentifierStructure;

import sun.security.x509.Extension;

import br.ufsc.jpki.ca.CertificationAuthority.AuthKeyIdType;
import br.ufsc.jpki.container.CertParameters;
import br.ufsc.jpki.exception.CertificationAuthorityBuildingException;
import br.ufsc.jpki.io.FileHandler;
import br.ufsc.jpki.provider.CryptographicServicesProvider;
import static br.ufsc.jpki.provider.CryptographicServicesProvider.KeyType;

public class CertificationAuthorityBuilder {

private String label;

private int caLevel;

private X509Certificate certificate;

private KeyPair keyPair;

private CertificationAuthority issuingCA;

private CryptographicServicesProvider csp;

private CertParameters certParameters;

public CertificationAuthorityBuilder() {
this(null);
}

public CertificationAuthorityBuilder(CryptographicServicesProvider provider) {
this.csp = provider;
}

public void setLabel(String label) {
this.label = label;
}

public void setCryptographicServiceProvider(
CryptographicServicesProvider provider) {
this.csp = provider;
}
```

```
public CertificationAuthority build()
throws CertificationAuthorityBuildingException {
this.generateKeyPair(KeyType.RSA);
if (this.certParameters != null
&& this.certParameters.getNotBefore() != null
&& this.certParameters.getNotAfter() != null
&& this.certParameters.getSubject() != null) {
this.generateCACertificate();
}
if (this.checkReadiness()) {
return new CertificationAuthority(this.label, this.caLevel,
this.certificate, this.keyPair.getPrivate(), this
.getAuthKeyIdType());
} else {
throw new CertificationAuthorityBuildingException(
"Error in CA building, missing arguments.");
}
}

public void setCertParameters(CertParameters c) {
this.certParameters = c;
}

// public CertificationAuthority buildFromRequest(CertificationRequest req)
// throws CertificationAuthorityBuildingException {
// this.generateKeyPair(KeyType.RSA);
// this.generateSelfSignedCertificate(req);
// if (this.checkReadiness()) {
// return new CertificationAuthority(this.label, this.certificate,
// this.keyPair.getPrivate());
// }
// else {
// throw new CertificationAuthorityBuildingException("Error in CA building,
// missing arguments.");
// }
// }

private void generateKeyPair(KeyType type)
throws CertificationAuthorityBuildingException {
if (this.csp != null && this.keyPair == null) {
try {
this.keyPair = this.csp.generateKeyPair(type);
} catch (NoSuchAlgorithmException e) {
throw new CertificationAuthorityBuildingException(
```

```
"Error creating CA key pair");
}
} else {
throw new CertificationAuthorityBuildingException(
"A CryptographicServiceProvider was not found.");
}
}

private void generateCACertificate()
throws CertificationAuthorityBuildingException {
try {
X509V3CertificateGenerator certGenerator = new X509V3CertificateGenerator();
X500Principal caSubject = new X500Principal(this.certParameters
.getSubject());
PrivateKey issuerPrivKey = null;
BigInteger serialNumber = null;
certGenerator.setSubjectDN(caSubject);
if (this.issuingCA == null) {
this.caLevel = 0;
certGenerator.setIssuerDN(caSubject);
issuerPrivKey = this.keyPair.getPrivate();
serialNumber = new BigInteger("1");
} else {
this.caLevel = this.issuingCA.getLevel() + 1;
certGenerator.addExtension(
X509Extensions.AuthorityKeyIdentifier, false, this
.buildAuthorityKeyIdentifier());
certGenerator.setIssuerDN(this.issuingCA.getCertificate()
.getSubjectX500Principal());
issuerPrivKey = this.issuingCA.getPrivateKey();
serialNumber = this.issuingCA.getNextCertSerial();
}
certGenerator.addExtension(X509Extensions.SubjectKeyIdentifier,
false, this.buildSubjectKeyIdentifier());
certGenerator.setNotBefore(this.certParameters.getNotBefore()
.getTime());
certGenerator.setNotAfter(this.certParameters.getNotAfter()
.getTime());
certGenerator.setPublicKey(this.keyPair.getPublic());
certGenerator.setSerialNumber(serialNumber);
certGenerator.setSignatureAlgorithm("SHA1withRSA");
if (this.certParameters.getPolicyOID() != null) {
ASN1Sequence certPolicies = this.buildCertPolicies(
this.certParameters.getPolicyOID(), this.certParameters
```

```
.getCpsURI());
certGenerator.addExtension(X509Extensions.CertificatePolicies,
false, certPolicies);
}
if (this.certParameters.getCrlDP() != null) {
CRLDistPoint crlDistPoints = this
.buildCRLDistPointExtension(this.certParameters
.getCrlDP());
certGenerator.addExtension(
X509Extensions.CRLDistributionPoints, false,
crlDistPoints);
}
if (this.certParameters.getKeyUsages() != null) {
KeyUsage keyUsage = this
.buildKeyUsageExtension(this.certParameters
.getKeyUsages());
certGenerator.addExtension(X509Extensions.KeyUsage, true,
keyUsage);
}
if (this.certParameters.getPathLen() != null) {
certGenerator.addExtension(X509Extensions.BasicConstraints,
true, new BasicConstraints(Integer
.parseInt(this.certParameters.getPathLen())));
} else {
certGenerator.addExtension(X509Extensions.BasicConstraints,
true, new BasicConstraints(true));
}
this.certificate = certGenerator
.generateX509Certificate(issuerPrivKey);
} catch (Exception ex) {
throw new CertificationAuthorityBuildingException(ex
.getLocalizedMessage(), ex);
}
}

private ASN1Sequence buildCertPolicies(String oid, String cpsURI) {
CertPolicyId policyIdentifier = new CertPolicyId(this.certParameters
.getPolicyOID());
ASN1Sequence qualifiers = null;
if (cpsURI != null) {
PolicyQualifierInfo pqi = new PolicyQualifierInfo(cpsURI);
qualifiers = new DERSequence(pqi);
return new DERSequence(new PolicyInformation(policyIdentifier,
qualifiers));
```

```
}
return new DERSequence(new PolicyInformation(policyIdentifier));
}

private CRLDistPoint buildCRLDistPointExtension(String uri) {
GeneralName crlUri = new GeneralName(
GeneralName.uniformResourceIdentifier, uri);
DistributionPointName distPointName = new DistributionPointName(
DistributionPointName.FULL_NAME, crlUri);
DistributionPoint[] distPoints = new DistributionPoint[1];
distPoints[0] = new DistributionPoint(distPointName, null, null);
return new CRLDistPoint(distPoints);
}

private KeyUsage buildKeyUsageExtension(
Vector<CertParameters.KeyPurpose> keyUsages) {
int keyUsageMask = 0;
if (keyUsages.contains(CertParameters.KeyPurpose.cRLSign)) {
keyUsageMask |= KeyUsage.cRLSign;
}
if (keyUsages.contains(CertParameters.KeyPurpose.dataEncipherment)) {
keyUsageMask |= KeyUsage.dataEncipherment;
}
if (keyUsages.contains(CertParameters.KeyPurpose.decipherOnly)) {
keyUsageMask |= KeyUsage.dataEncipherment;
}
if (keyUsages.contains(CertParameters.KeyPurpose.digitalSignature)) {
keyUsageMask |= KeyUsage.digitalSignature;
}
if (keyUsages.contains(CertParameters.KeyPurpose.encipherOnly)) {
keyUsageMask |= KeyUsage.encipherOnly;
}
if (keyUsages.contains(CertParameters.KeyPurpose.keyAgreement)) {
keyUsageMask |= KeyUsage.keyAgreement;
}
if (keyUsages.contains(CertParameters.KeyPurpose.keyCertSign)) {
keyUsageMask |= KeyUsage.keyCertSign;
}
if (keyUsages.contains(CertParameters.KeyPurpose.keyEncipherment)) {
keyUsageMask |= KeyUsage.dataEncipherment;
}
if (keyUsages.contains(CertParameters.KeyPurpose.nonRepudiation)) {
keyUsageMask |= KeyUsage.nonRepudiation;
}
```

```
return new KeyUsage(keyUsageMask);
}

private SubjectKeyIdentifier buildSubjectKeyIdentifier()
throws CertificateParsingException {
return new SubjectKeyIdentifierStructure(this.keyPair.getPublic());
}

private AuthorityKeyIdentifier buildAuthorityKeyIdentifier()
throws InvalidKeyException, CertificateException {
CertificationAuthority.AuthKeyIdType authKeyIdType = this
.getAuthKeyIdType();
// if (authKeyIdType == AuthKeyIdType.COMPLETE) {
// return new
// AuthorityKeyIdentifierStructure(this.issuingCA.getCertificate());
// }
// else {
DEREncodable keyidentifier = new SubjectKeyIdentifierStructure(
this.issuingCA.getCertificate().getPublicKey());
ASN1EncodableVector v = new ASN1EncodableVector();
v.add(new DERTaggedObject(false, 0, keyidentifier));
if (authKeyIdType != AuthKeyIdType.BASIC) {
if (authKeyIdType != AuthKeyIdType.NODIR) {
GeneralNames gns = new GeneralNames(new GeneralName(
PrincipalUtil.getIssuerX509Principal(this.issuingCA
.getCertificate())));
v.add(new DERTaggedObject(false, 1, gns));
}
if (authKeyIdType != AuthKeyIdType.NOSERIAL) {
v.add(new DERTaggedObject(false, 2, new DERInteger(
this.issuingCA.getCertificate().getSerialNumber())));
}
}
return new AuthorityKeyIdentifier(new DERSequence(v));
// }
}

private boolean checkReadiness() {
if (this.label != null && this.certificate != null
&& this.keyPair != null && this.csp != null) {
return true;
}
return false;
}
```

```
public void setIssuingCA(CertificationAuthority ica) {
this.issuingCA = ica;
}

public CertificationAuthority.AuthKeyIdType getAuthKeyIdType() {
CertificationAuthority.AuthKeyIdType authKeyIdType = CertificationAuthority.AuthKeyIdType.BASIC;
if ("nodir".equals(this.certParameters.getAuthKeyIdType())) {
authKeyIdType = CertificationAuthority.AuthKeyIdType.NODIR;
} else if ("noserial".equals(this.certParameters.getAuthKeyIdType())) {
authKeyIdType = CertificationAuthority.AuthKeyIdType.NOSERIAL;
} else if ("complete".equals(this.certParameters.getAuthKeyIdType())) {
authKeyIdType = CertificationAuthority.AuthKeyIdType.COMPLETE;
}
return authKeyIdType;
}

}
package br.ufsc.jpki;

import br.ufsc.jpki.persistence.HibernateUtil;

public class Main {

/**
 * @param args
 */
public static void main(String[] args) {
int status = 0;
try {
PKIManager manager = PKIManager.getInstance();
if (args.length >= 1) {
manager.executeTask(args);
HibernateUtil.shutdown();
} else {
manager.printHelp();
status = 1;
}
} catch (Exception e) {
e.printStackTrace();
System.err.println("The following error was returned");
System.err.println("Error: " + e.getLocalizedMessage());
status = 1;
}
```

```
System.exit(status);
}
}
package br.ufsc.jpki.provider;

import java.io.IOException;
import java.io.InputStream;

import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.PasswordCallback;
import javax.security.auth.callback.UnsupportedCallbackException;

public class PKCS11CallbackHandler implements CallbackHandler {

public void handle(Callback[] callbacks) throws IOException,
UnsupportedCallbackException {
for (int i = 0; i < callbacks.length; i++) {
if (callbacks[i] instanceof PasswordCallback) {
PasswordCallback pwdCallback = (PasswordCallback) callbacks[i];
System.err.print(pwdCallback.getPrompt());
pwdCallback.setPassword(this.readPassword(System.in));
} else {
throw new UnsupportedCallbackException(callbacks[i],
"Unrecognized Callback");
}
}
}

private char[] readPassword(InputStream in) throws IOException {
byte[] pin = new byte[8];
int nread = in.read(pin);
String pinText = "";
while (nread > 0) {
if (Character.isDigit(pin[nread - 1])) {
pinText = (char) pin[nread - 1] + pinText;
}
nread--;
}
return pinText.toCharArray();
}
}
package br.ufsc.jpki.provider;
```

```
import java.io.File;
import java.io.FileInputStream;
import java.math.BigInteger;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.Security;
import java.util.HashMap;
import java.util.Set;

import org.bouncycastle.crypto.AsymmetricCipherKeyPair;
import org.bouncycastle.crypto.AsymmetricCipherKeyPairGenerator;
import org.bouncycastle.crypto.generators.RSAKeyPairGenerator;
import org.bouncycastle.crypto.params.DSAKeyGenerationParameters;
import org.bouncycastle.crypto.params.RSAKeyGenerationParameters;
import org.bouncycastle.jce.provider.BouncyCastleProvider;

public class CryptographicServicesProvider {

private KeyPairGenerator kpGenerator;

public enum KeyType {
RSA, DSA
}

private HashMap<KeyType, String> algorithms;

public CryptographicServicesProvider() {
Security.addProvider(new BouncyCastleProvider());
this.algorithms = new HashMap<KeyType, String>();
this.algorithms.put(KeyType.RSA, "RSA");
this.algorithms.put(KeyType.DSA, "DSA");
}

public KeyPair generateKeyPair(KeyType keyAlgorithm)
throws NoSuchAlgorithmException {
KeyPair keyPair = null;
if (this.kpGenerator == null) {
this.kpGenerator = KeyPairGenerator.getInstance(this.algorithms
.get(keyAlgorithm));
this.kpGenerator.initialize(1024, this.generateRandom());
}
keyPair = this.kpGenerator.generateKeyPair();
```

```
return keyPair;
}

public SecureRandom generateRandom() {
SecureRandom rand = null;
try {
rand = SecureRandom.getInstance("SHA1PRNG", "SUN");
if (new File("/dev/urandom").exists()) {
byte[] salt = new byte[8192];
new FileInputStream("/dev/urandom").read(salt);
rand.setSeed(salt);
}
} catch (Exception ex) {
ex.printStackTrace();
}
return rand;
}
}
package br.ufsc.jpki.provider;

import java.math.BigInteger;
import java.security.GeneralSecurityException;
import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.PrivateKey;
import java.security.Provider;
import java.security.SecureRandom;
import java.security.Security;
import java.security.Signature;
import java.security.Provider.Service;
import java.security.cert.X509Certificate;
import java.security.interfaces.RSAPrivateKey;
import java.util.Calendar;
import java.util.Date;
import java.util.Enumeration;
import java.util.Iterator;
import java.util.Set;

import javax.security.auth.x500.X500Principal;

import org.bouncycastle.jce.cert.CertificateFactory;
```

```
import org.bouncycastle.jce.provider.BouncyCastleProvider;
import org.bouncycastle.x509.X509V3CertificateGenerator;

import br.ufsc.jpki.PKIManager;
import br.ufsc.jpki.exception.PKCS11Exception;

import sun.security.pkcs11.SunPKCS11;

public class PKCS11Handler {

private Provider provider;

private final String pkcs11ConfigFile = "pkcs11.cfg";

private KeyStore.Builder keyStoreBuilder = null;

public PKCS11Handler() throws PKCS11Exception {
this.provider = new SunPKCS11(pkcs11ConfigFile);
Security.addProvider(this.provider);
KeyStore.CallbackHandlerProtection protection = null;
// try {
// protection = new
// KeyStore.CallbackHandlerProtection(PKIManager.getInstance(null));
protection = new KeyStore.CallbackHandlerProtection(
new PKCS11CallbackHandler());
// }
// catch (InstantiationException e) {
// throw new PKCS11Exception("Callback handler not available", e);
// }
this.keyStoreBuilder = KeyStore.Builder.newInstance("PKCS11",
this.provider, protection);
}

private KeyStore loadPKCS11KeyStore() throws PKCS11Exception {
try {
return this.keyStoreBuilder.getKeyStore();
} catch (Exception e) {
throw new PKCS11Exception(e.getLocalizedMessage(), e);
}
}

public void generatePrivateKey() throws PKCS11Exception {
KeyStore ks = this.loadPKCS11KeyStore();
System.out.println(ks.getProvider().getServices());
```

```
Provider prov = new BouncyCastleProvider();
Security.addProvider(new BouncyCastleProvider());
try {
KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
kpg.initialize(1024);
KeyPair kp = kpg.generateKeyPair();
X509V3CertificateGenerator certGen = new X509V3CertificateGenerator();
certGen.setSubjectDN(new X500Principal(
"CN=Operador O=ICP-Brasil OU=ITI"));
Calendar notBefore = Calendar.getInstance();
certGen.setNotBefore(notBefore.getTime());
notBefore.add(Calendar.YEAR, 5);
certGen.setNotAfter(notBefore.getTime());
certGen.setSerialNumber(new BigInteger("1"));
certGen.setIssuerDN(new X500Principal(
"CN=Operador O=ICP-Brasil OU=ITI"));
certGen.setSignatureAlgorithm("SHA1withRSA");
certGen.setPublicKey(kp.getPublic());
X509Certificate cert = certGen.generateX509Certificate(kp
.getPrivate());
X509Certificate[] chain = new X509Certificate[1];
chain[0] = cert;
ks.setKeyEntry("operador.key", kp.getPrivate().getEncoded(), chain);
ks.store(null);
} catch (Exception e) {
e.printStackTrace();
throw new PKCS11Exception(e);
}
}

public Enumeration<String> getAvailableCertificates()
throws PKCS11Exception {
KeyStore ks = this.loadPKCS11KeyStore();
try {
return ks.aliases();
} catch (KeyStoreException e) {
throw new PKCS11Exception(e.getLocalizedMessage(), e);
}
}

public PrivateKey loadPrivateKey(String alias) throws PKCS11Exception {
KeyStore ks = this.loadPKCS11KeyStore();
PrivateKey privateKey = null;
try {
```

```
privateKey = (PrivateKey) ks.getKey(alias, null);
} catch (Exception e) {
throw new PKCS11Exception(e.getLocalizedMessage(), e);
}
return privateKey;
}

public X509Certificate loadCertificate(String alias)
throws PKCS11Exception, IllegalStateException {
KeyStore ks = this.loadPKCS11KeyStore();
X509Certificate certificate = null;
try {
certificate = (X509Certificate) ks.getCertificate(alias);
} catch (Exception e) {
throw new PKCS11Exception(e.getLocalizedMessage(), e);
}
return certificate;
}

public boolean provePossession(String alias) throws PKCS11Exception {
X509Certificate certificate = null;
PrivateKey pk = null;
double randNum = Math.random() * 1000000000;
String challenge = "<<<-random-number = " + randNum + "->>>";
try {
certificate = this.loadCertificate(alias);
pk = this.loadPrivateKey(alias);
} catch (Exception e) {
throw new PKCS11Exception(e.getLocalizedMessage(), e);
}
try {
Signature signatureAlgorithm = Signature.getInstance("SHA1withRSA",
this.provider);
signatureAlgorithm.initSign(pk);
signatureAlgorithm.update(challenge.getBytes());
byte[] digitalSignature = signatureAlgorithm.sign();
Signature signatureAlgorithm2 = Signature
.getInstance("SHA1withRSA");
signatureAlgorithm2.initVerify(certificate);
signatureAlgorithm2.update(challenge.getBytes());
return signatureAlgorithm2.verify(digitalSignature);
} catch (Exception e) {
throw new PKCS11Exception(e.getLocalizedMessage(), e);
}
```

```
}

public void storeCertificate(String alias, X509Certificate cert)
throws PKCS11Exception {
KeyStore ks = this.loadPKCS11KeyStore();
try {
if (!ks.containsAlias(alias)) {
ks.setCertificateEntry(alias, cert);
} else {
new PKCS11Exception("Alias already exists in key store");
}
} catch (GeneralSecurityException e) {
throw new PKCS11Exception(e.getLocalizedMessage(), e);
}
}
}
/**
 * <p>Encodes and decodes to and from Base64 notation.</p>
 * <p>Homepage: <a href="http://iharder.net/base64">http://iharder.net/base64</a>.</p>
 *
 * <p>
 * Change Log:
 * </p>
 * <ul>
 *  <li>v2.2.1 - Fixed bug using URL_SAFE and ORDERED encodings. Fixed bug
 *   when using very small files (~< 40 bytes).</li>
 *  <li>v2.2 - Added some helper methods for encoding/decoding directly from
 *   one file to the next. Also added a main() method to support command line
 *   encoding/decoding from one file to the next. Also added these Base64 dialects:
 *   <ol>
 *   <li>The default is RFC3548 format.</li>
 *   <li>Calling Base64.setFormat(Base64.BASE64_FORMAT.URLSAFE_FORMAT) generates
 *   URL and file name friendly format as described in Section 4 of RFC3548.
 *   http://www.faqs.org/rfcs/rfc3548.html</li>
 *   <li>Calling Base64.setFormat(Base64.BASE64_FORMAT.ORDERED_FORMAT) generates
 *   URL and file name friendly format that preserves lexical ordering as described
 *   in http://www.faqs.org/qa/rfcc-1940.html</li>
 *   </ol>
 *   Special thanks to Jim Kellerman at <a href="http://www.powerset.com/">http://www.powerset.com/</a>
 *   for contributing the new Base64 dialects.
 *  </li>
 *
 *  <li>v2.1 - Cleaned up javadoc comments and unused variables and methods. Added
 *   some convenience methods for reading and writing to and from files.</li>
```

```
 *  <li>v2.0.2 - Now specifies UTF-8 encoding in places where the code fails on systems
 *   with other encodings (like EBCDIC).</li>
 *  <li>v2.0.1 - Fixed an error when decoding a single byte, that is, when the
 *   encoded data was a single byte.</li>
 *  <li>v2.0 - I got rid of methods that used booleans to set options.
 *   Now everything is more consolidated and cleaner. The code now detects
 *   when data that's being decoded is gzip-compressed and will decompress it
 *   automatically. Generally things are cleaner. You'll probably have to
 *   change some method calls that you were making to support the new
 *   options format (<tt>int</tt>s that you "OR" together).</li>
 *  <li>v1.5.1 - Fixed bug when decompressing and decoding to a
 *   byte[] using <tt>decode( String s, boolean gzipCompressed )</tt>.
 *   Added the ability to "suspend" encoding in the Output Stream so
 *   you can turn on and off the encoding if you need to embed base64
 *   data in an otherwise "normal" stream (like an XML file).</li>
 *  <li>v1.5 - Output stream pases on flush() command but doesn't do anything itself.
 *      This helps when using GZIP streams.
 *      Added the ability to GZip-compress objects before encoding them.</li>
 *  <li>v1.4 - Added helper methods to read/write files.</li>
 *  <li>v1.3.6 - Fixed OutputStream.flush() so that 'position' is reset.</li>
 *  <li>v1.3.5 - Added flag to turn on and off line breaks. Fixed bug in input stream
 *      where last buffer being read, if not completely full, was not returned.</li>
 *  <li>v1.3.4 - Fixed when "improperly padded stream" error was thrown at the wrong time.</li>
 *  <li>v1.3.3 - Fixed I/O streams which were totally messed up.</li>
 * </ul>
 *
 * <p>
 * I am placing this code in the Public Domain. Do with it as you will.
 * This software comes with no guarantees or warranties but with
 * plenty of well-wishing instead!
 * Please visit <a href="http://iharder.net/base64">http://iharder.net/base64</a>
 * periodically to check for updates or to contribute improvements.
 * </p>
 *
 * @author Robert Harder
 * @author rob@iharder.net
 * @version 2.2.1
 */

package br.ufsc.jpki.util;

public class Base64 {

/* ******** P U B L I C F I E L D S ******** */
```

```
/** No options specified. Value is zero. */
public final static int NO_OPTIONS = 0;

/** Specify encoding. */
public final static int ENCODE = 1;

/** Specify decoding. */
public final static int DECODE = 0;

/** Specify that data should be gzip-compressed. */
public final static int GZIP = 2;

/** Don’t break lines when encoding (violates strict Base64 specification) */
public final static int DONT_BREAK_LINES = 8;

/**
 * Encode using Base64-like encoding that is URL- and Filename-safe as
 * described in Section 4 of RFC3548: <a
 * href="http://www.faqs.org/rfcs/rfc3548.html">http://www.faqs.org/rfcs/rfc3548.html</a>.
 * It is important to note that data encoded this way is <em>not</em>
 * officially valid Base64, or at the very least should not be called Base64
 * without also specifying that is was encoded using the URL- and
 * Filename-safe dialect.
 */
public final static int URL_SAFE = 16;

/**
 * Encode using the special "ordered" dialect of Base64 described here: <a
 * href="http://www.faqs.org/qa/rfcc-1940.html">http://www.faqs.org/qa/rfcc-1940.html</a>.
 */
public final static int ORDERED = 32;

/* ******** P R I V A T E F I E L D S ******** */

/** Maximum line length (76) of Base64 output. */
private final static int MAX_LINE_LENGTH = 76;

/** The equals sign (=) as a byte. */
private final static byte EQUALS_SIGN = (byte) ’=’;

/** The new line character (\n) as a byte. */
private final static byte NEW_LINE = (byte) ’\n’;
```

```
/** Preferred encoding. */
private final static String PREFERRED_ENCODING = "UTF-8";

// I think I end up not using the BAD_ENCODING indicator.
// private final static byte BAD_ENCODING = -9; // Indicates error in
// encoding
private final static byte WHITE_SPACE_ENC = -5; // Indicates white space in
// encoding

private final static byte EQUALS_SIGN_ENC = -1; // Indicates equals sign in
// encoding

/* ******** S T A N D A R D B A S E 6 4 A L P H A B E T ******** */

/** The 64 valid Base64 values. */
// private final static byte[] ALPHABET;
/*
 * Host platform me be something funny like EBCDIC, so we hardcode these
 * values.
 */
private final static byte[] _STANDARD_ALPHABET = { (byte) 'A', (byte) 'B',
(byte) 'C', (byte) 'D', (byte) 'E', (byte) 'F', (byte) 'G',
(byte) 'H', (byte) 'I', (byte) 'J', (byte) 'K', (byte) 'L',
(byte) 'M', (byte) 'N', (byte) 'O', (byte) 'P', (byte) 'Q',
(byte) 'R', (byte) 'S', (byte) 'T', (byte) 'U', (byte) 'V',
(byte) 'W', (byte) 'X', (byte) 'Y', (byte) 'Z', (byte) 'a',
(byte) 'b', (byte) 'c', (byte) 'd', (byte) 'e', (byte) 'f',
(byte) 'g', (byte) 'h', (byte) 'i', (byte) 'j', (byte) 'k',
(byte) 'l', (byte) 'm', (byte) 'n', (byte) 'o', (byte) 'p',
(byte) 'q', (byte) 'r', (byte) 's', (byte) 't', (byte) 'u',
(byte) 'v', (byte) 'w', (byte) 'x', (byte) 'y', (byte) 'z',
(byte) '0', (byte) '1', (byte) '2', (byte) '3', (byte) '4',
(byte) '5', (byte) '6', (byte) '7', (byte) '8', (byte) '9',
(byte) '+', (byte) '/' };

/**
 * Translates a Base64 value to either its 6-bit reconstruction value or a
 * negative number indicating some other meaning.
 */
private final static byte[] _STANDARD_DECODABET = { -9, -9, -9, -9, -9, -9,
-9, -9, -9, // Decimal 0 - 8
-5, -5, // Whitespace: Tab and Linefeed
-9, -9, // Decimal 11 - 12
-5, // Whitespace: Carriage Return
```

```
-9, -9, -9, -9, -9, -9, -9, -9, -9, -9, -9, -9, -9, // Decimal 14 -
// 26
-9, -9, -9, -9, -9, // Decimal 27 - 31
-5, // Whitespace: Space
-9, -9, -9, -9, -9, -9, -9, -9, -9, -9, // Decimal 33 - 42
62, // Plus sign at decimal 43
-9, -9, -9, // Decimal 44 - 46
63, // Slash at decimal 47
52, 53, 54, 55, 56, 57, 58, 59, 60, 61, // Numbers zero through nine
-9, -9, -9, // Decimal 58 - 60
-1, // Equals sign at decimal 61
-9, -9, -9, // Decimal 62 - 64
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, // Letters 'A'
// through 'N'
14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, // Letters 'O'
// through 'Z'
-9, -9, -9, -9, -9, -9, // Decimal 91 - 96
26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, // Letters 'a'
// through 'm'
39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, // Letters 'n'
// through 'z'
-9, -9, -9, -9 // Decimal 123 - 126
/*
 * ,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 127 - 139
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 140 - 152
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 153 - 165
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 166 - 178
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 179 - 191
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 192 - 204
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 205 - 217
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 218 - 230
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 231 - 243
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9 // Decimal 244 - 255
 */
};

/* ******** U R L S A F E B A S E 6 4 A L P H A B E T ******** */

/**
 * Used in the URL- and Filename-safe dialect described in Section 4 of
 * RFC3548: <a
 * href="http://www.faqs.org/rfcs/rfc3548.html">http://www.faqs.org/rfcs/rfc3548.html</a>.
 * Notice that the last two bytes become "hyphen" and "underscore" instead
 * of "plus" and "slash."
```

```
 */
private final static byte[] _URL_SAFE_ALPHABET = { (byte) 'A', (byte) 'B',
(byte) 'C', (byte) 'D', (byte) 'E', (byte) 'F', (byte) 'G',
(byte) 'H', (byte) 'I', (byte) 'J', (byte) 'K', (byte) 'L',
(byte) 'M', (byte) 'N', (byte) 'O', (byte) 'P', (byte) 'Q',
(byte) 'R', (byte) 'S', (byte) 'T', (byte) 'U', (byte) 'V',
(byte) 'W', (byte) 'X', (byte) 'Y', (byte) 'Z', (byte) 'a',
(byte) 'b', (byte) 'c', (byte) 'd', (byte) 'e', (byte) 'f',
(byte) 'g', (byte) 'h', (byte) 'i', (byte) 'j', (byte) 'k',
(byte) 'l', (byte) 'm', (byte) 'n', (byte) 'o', (byte) 'p',
(byte) 'q', (byte) 'r', (byte) 's', (byte) 't', (byte) 'u',
(byte) 'v', (byte) 'w', (byte) 'x', (byte) 'y', (byte) 'z',
(byte) '0', (byte) '1', (byte) '2', (byte) '3', (byte) '4',
(byte) '5', (byte) '6', (byte) '7', (byte) '8', (byte) '9',
(byte) '-', (byte) '_' };

/**
 * Used in decoding URL- and Filename-safe dialects of Base64.
 */
private final static byte[] _URL_SAFE_DECODABET = { -9, -9, -9, -9, -9, -9,
-9, -9, -9, // Decimal 0 - 8
-5, -5, // Whitespace: Tab and Linefeed
-9, -9, // Decimal 11 - 12
-5, // Whitespace: Carriage Return
-9, -9, -9, -9, -9, -9, -9, -9, -9, -9, -9, -9, -9, // Decimal 14 -
// 26
-9, -9, -9, -9, -9, // Decimal 27 - 31
-5, // Whitespace: Space
-9, -9, -9, -9, -9, -9, -9, -9, -9, -9, // Decimal 33 - 42
-9, // Plus sign at decimal 43
-9, // Decimal 44
62, // Minus sign at decimal 45
-9, // Decimal 46
-9, // Slash at decimal 47
52, 53, 54, 55, 56, 57, 58, 59, 60, 61, // Numbers zero through nine
-9, -9, -9, // Decimal 58 - 60
-1, // Equals sign at decimal 61
-9, -9, -9, // Decimal 62 - 64
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, // Letters 'A'
// through 'N'
14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, // Letters 'O'
// through 'Z'
-9, -9, -9, -9, // Decimal 91 - 94
63, // Underscore at decimal 95
```

```
-9, // Decimal 96
26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, // Letters ’a’
// through ’m’
39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, // Letters ’n’
// through ’z’
-9, -9, -9, -9 // Decimal 123 - 126
/*
 * ,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 127 - 139
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 140 - 152
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 153 - 165
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 166 - 178
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 179 - 191
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 192 - 204
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 205 - 217
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 218 - 230
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 231 - 243
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9 // Decimal 244 - 255
 */
};

/* ******** O R D E R E D B A S E 6 4 A L P H A B E T ******** */

/**
 * I don’t get the point of this technique, but it is described here: <a
 * href="http://www.faqs.org/qa/rfcc-1940.html">http://www.faqs.org/qa/rfcc-1940.html</a>.
 */
private final static byte[] _ORDERED_ALPHABET = { (byte) ’-’, (byte) ’0’,
(byte) ’1’, (byte) ’2’, (byte) ’3’, (byte) ’4’, (byte) ’5’,
(byte) ’6’, (byte) ’7’, (byte) ’8’, (byte) ’9’, (byte) ’A’,
(byte) ’B’, (byte) ’C’, (byte) ’D’, (byte) ’E’, (byte) ’F’,
(byte) ’G’, (byte) ’H’, (byte) ’I’, (byte) ’J’, (byte) ’K’,
(byte) ’L’, (byte) ’M’, (byte) ’N’, (byte) ’O’, (byte) ’P’,
(byte) ’Q’, (byte) ’R’, (byte) ’S’, (byte) ’T’, (byte) ’U’,
(byte) ’V’, (byte) ’W’, (byte) ’X’, (byte) ’Y’, (byte) ’Z’,
(byte) ’_’, (byte) ’a’, (byte) ’b’, (byte) ’c’, (byte) ’d’,
(byte) ’e’, (byte) ’f’, (byte) ’g’, (byte) ’h’, (byte) ’i’,
(byte) ’j’, (byte) ’k’, (byte) ’l’, (byte) ’m’, (byte) ’n’,
(byte) ’o’, (byte) ’p’, (byte) ’q’, (byte) ’r’, (byte) ’s’,
(byte) ’t’, (byte) ’u’, (byte) ’v’, (byte) ’w’, (byte) ’x’,
(byte) ’y’, (byte) ’z’ };

/**
 * Used in decoding the "ordered" dialect of Base64.
 */
```

```
private final static byte[] _ORDERED_DECODABET = { -9, -9, -9, -9, -9, -9,
-9, -9, -9, // Decimal 0 - 8
-5, -5, // Whitespace: Tab and Linefeed
-9, -9, // Decimal 11 - 12
-5, // Whitespace: Carriage Return
-9, -9, -9, -9, -9, -9, -9, -9, -9, -9, -9, -9, -9, // Decimal 14 -
// 26
-9, -9, -9, -9, -9, // Decimal 27 - 31
-5, // Whitespace: Space
-9, -9, -9, -9, -9, -9, -9, -9, -9, -9, // Decimal 33 - 42
-9, // Plus sign at decimal 43
-9, // Decimal 44
0, // Minus sign at decimal 45
-9, // Decimal 46
-9, // Slash at decimal 47
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, // Numbers zero through nine
-9, -9, -9, // Decimal 58 - 60
-1, // Equals sign at decimal 61
-9, -9, -9, // Decimal 62 - 64
11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, // Letters 'A'
// through 'M'
24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, // Letters 'N'
// through 'Z'
-9, -9, -9, -9, // Decimal 91 - 94
37, // Underscore at decimal 95
-9, // Decimal 96
38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, // Letters 'a'
// through 'm'
51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, // Letters 'n'
// through 'z'
-9, -9, -9, -9 // Decimal 123 - 126
/*
 * ,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 127 - 139
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 140 - 152
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 153 - 165
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 166 - 178
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 179 - 191
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 192 - 204
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 205 - 217
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 218 - 230
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9, // Decimal 231 - 243
 * -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9 // Decimal 244 - 255
 */
};
```

```
/* ******** D E T E R M I N E W H I C H A L H A B E T ******** */

/**
 * Returns one of the _SOMETHING_ALPHABET byte arrays depending on the
 * options specified. It's possible, though silly, to specify ORDERED and
 * URLSAFE in which case one of them will be picked, though there is no
 * guarantee as to which one will be picked.
 */
private final static byte[] getAlphabet(int options) {
if ((options & URL_SAFE) == URL_SAFE)
return _URL_SAFE_ALPHABET;
else if ((options & ORDERED) == ORDERED)
return _ORDERED_ALPHABET;
else
return _STANDARD_ALPHABET;

} // end getAlphabet

/**
 * Returns one of the _SOMETHING_DECODABET byte arrays depending on the
 * options specified. It's possible, though silly, to specify ORDERED and
 * URL_SAFE in which case one of them will be picked, though there is no
 * guarantee as to which one will be picked.
 */
private final static byte[] getDecodabet(int options) {
if ((options & URL_SAFE) == URL_SAFE)
return _URL_SAFE_DECODABET;
else if ((options & ORDERED) == ORDERED)
return _ORDERED_DECODABET;
else
return _STANDARD_DECODABET;

} // end getAlphabet

/** Defeats instantiation. */
private Base64() {
}

/**
 * Encodes or decodes two files from the command line; <strong>feel free to
 * delete this method (in fact you probably should) if you're embedding this
 * code into a larger program.</strong>
 */
```

```
public final static void main(String[] args) {
if (args.length < 3) {
usage("Not enough arguments.");
} // end if: args.length < 3
else {
String flag = args[0];
String infile = args[1];
String outfile = args[2];
if (flag.equals("-e")) {
Base64.encodeFileToFile(infile, outfile);
} // end if: encode
else if (flag.equals("-d")) {
Base64.decodeFileToFile(infile, outfile);
} // end else if: decode
else {
usage("Unknown flag: " + flag);
} // end else
} // end else
} // end main

/**
 * Prints command line usage.
 *
 * @param msg
 *            A message to include with usage info.
 */
private final static void usage(String msg) {
System.err.println(msg);
System.err.println("Usage: java Base64 -e|-d inputfile outputfile");
} // end usage

/* ******** E N C O D I N G M E T H O D S ******** */

/**
 * Encodes up to the first three bytes of array <var>threeBytes</var> and
 * returns a four-byte array in Base64 notation. The actual number of
 * significant bytes in your array is given by <var>numSigBytes</var>. The
 * array <var>threeBytes</var> needs only be as big as <var>numSigBytes</var>.
 * Code can reuse a byte array by passing a four-byte array as <var>b4</var>.
 *
 * @param b4
 *            A reusable byte array to reduce array instantiation
 * @param threeBytes
 *            the array to convert
```

```
 * @param numSigBytes
 *            the number of significant bytes in your array
 * @return four byte array in Base64 notation.
 * @since 1.5.1
 */
private static byte[] encode3to4(byte[] b4, byte[] threeBytes,
int numSigBytes, int options) {
encode3to4(threeBytes, 0, numSigBytes, b4, 0, options);
return b4;
} // end encode3to4

/**
 * <p>
 * Encodes up to three bytes of the array <var>source</var> and writes the
 * resulting four Base64 bytes to <var>destination</var>. The source and
 * destination arrays can be manipulated anywhere along their length by
 * specifying <var>srcOffset</var> and <var>destOffset</var>. This method
 * does not check to make sure your arrays are large enough to accomodate
 * <var>srcOffset</var> + 3 for the <var>source</var> array or
 * <var>destOffset</var> + 4 for the <var>destination</var> array. The
 * actual number of significant bytes in your array is given by
 * <var>numSigBytes</var>.
 * </p>
 * <p>
 * This is the lowest level of the encoding methods with all possible
 * parameters.
 * </p>
 *
 * @param source
 *            the array to convert
 * @param srcOffset
 *            the index where conversion begins
 * @param numSigBytes
 *            the number of significant bytes in your array
 * @param destination
 *            the array to hold the conversion
 * @param destOffset
 *            the index where output will be put
 * @return the <var>destination</var> array
 * @since 1.3
 */
private static byte[] encode3to4(byte[] source, int srcOffset,
int numSigBytes, byte[] destination, int destOffset, int options) {
byte[] ALPHABET = getAlphabet(options);
```

```
// 1 2 3
// 01234567890123456789012345678901 Bit position
// --------000000001111111122222222 Array position from threeBytes
// --------| || || || | Six bit groups to index ALPHABET
// >>18 >>12 >> 6 >> 0 Right shift necessary
// 0x3f 0x3f 0x3f Additional AND

// Create buffer with zero-padding if there are only one or two
// significant bytes passed in the array.
// We have to shift left 24 in order to flush out the 1's that appear
// when Java treats a value as negative that is cast from a byte to an
// int.
int inBuff = (numSigBytes > 0 ? ((source[srcOffset] << 24) >>> 8) : 0)
| (numSigBytes > 1 ? ((source[srcOffset + 1] << 24) >>> 16) : 0)
| (numSigBytes > 2 ? ((source[srcOffset + 2] << 24) >>> 24) : 0);

switch (numSigBytes) {
case 3:
destination[destOffset] = ALPHABET[(inBuff >>> 18)];
destination[destOffset + 1] = ALPHABET[(inBuff >>> 12) & 0x3f];
destination[destOffset + 2] = ALPHABET[(inBuff >>> 6) & 0x3f];
destination[destOffset + 3] = ALPHABET[(inBuff) & 0x3f];
return destination;

case 2:
destination[destOffset] = ALPHABET[(inBuff >>> 18)];
destination[destOffset + 1] = ALPHABET[(inBuff >>> 12) & 0x3f];
destination[destOffset + 2] = ALPHABET[(inBuff >>> 6) & 0x3f];
destination[destOffset + 3] = EQUALS_SIGN;
return destination;

case 1:
destination[destOffset] = ALPHABET[(inBuff >>> 18)];
destination[destOffset + 1] = ALPHABET[(inBuff >>> 12) & 0x3f];
destination[destOffset + 2] = EQUALS_SIGN;
destination[destOffset + 3] = EQUALS_SIGN;
return destination;

default:
return destination;
} // end switch
} // end encode3to4
```

```
/**
 * Serializes an object and returns the Base64-encoded version of that
 * serialized object. If the object cannot be serialized or there is another
 * error, the method will return <tt>null</tt>. The object is not
 * GZip-compressed before being encoded.
 *
 * @param serializableObject
 *            The object to encode
 * @return The Base64-encoded object
 * @since 1.4
 */
public static String encodeObject(java.io.Serializable serializableObject) {
return encodeObject(serializableObject, NO_OPTIONS);
} // end encodeObject

/**
 * Serializes an object and returns the Base64-encoded version of that
 * serialized object. If the object cannot be serialized or there is another
 * error, the method will return <tt>null</tt>.
 * <p>
 * Valid options:
 *
 * <pre>
 *   GZIP: gzip-compresses object before encoding it.
 *   DONT_BREAK_LINES: don't break lines at 76 characters
 *     &lt;i&gt;Note: Technically, this makes your encoding non-compliant.&lt;/i&gt;
 * </pre>
 *
 * <p>
 * Example: <code>encodeObject( myObj, Base64.GZIP )</code> or
 * <p>
 * Example:
 * <code>encodeObject( myObj, Base64.GZIP | Base64.DONT_BREAK_LINES )</code>
 *
 * @param serializableObject
 *            The object to encode
 * @param options
 *            Specified options
 * @return The Base64-encoded object
 * @see Base64#GZIP
 * @see Base64#DONT_BREAK_LINES
 * @since 2.0
 */
public static String encodeObject(java.io.Serializable serializableObject,
```

```
int options) {
// Streams
java.io.ByteArrayOutputStream baos = null;
java.io.OutputStream b64os = null;
java.io.ObjectOutputStream oos = null;
java.util.zip.GZIPOutputStream gzos = null;

// Isolate options
int gzip = (options & GZIP);
int dontBreakLines = (options & DONT_BREAK_LINES);

try {
// ObjectOutputStream -> (GZIP) -> Base64 -> ByteArrayOutputStream
baos = new java.io.ByteArrayOutputStream();
b64os = new Base64.OutputStream(baos, ENCODE | options);

// GZip?
if (gzip == GZIP) {
gzos = new java.util.zip.GZIPOutputStream(b64os);
oos = new java.io.ObjectOutputStream(gzos);
} // end if: gzip
else
oos = new java.io.ObjectOutputStream(b64os);

oos.writeObject(serializableObject);
} // end try
catch (java.io.IOException e) {
e.printStackTrace();
return null;
} // end catch
finally {
try {
oos.close();
} catch (Exception e) {
}
try {
gzos.close();
} catch (Exception e) {
}
try {
b64os.close();
} catch (Exception e) {
}
try {
```

```
baos.close();
} catch (Exception e) {
}
} // end finally

// Return value according to relevant encoding.
try {
return new String(baos.toByteArray(), PREFERRED_ENCODING);
} // end try
catch (java.io.UnsupportedEncodingException uue) {
return new String(baos.toByteArray());
} // end catch

} // end encode

/**
 * Encodes a byte array into Base64 notation. Does not GZip-compress data.
 *
 * @param source
 *            The data to convert
 * @since 1.4
 */
public static String encodeBytes(byte[] source) {
return encodeBytes(source, 0, source.length, NO_OPTIONS);
} // end encodeBytes

/**
 * Encodes a byte array into Base64 notation.
 * <p>
 * Valid options:
 *
 * <pre>
 *   GZIP: gzip-compresses object before encoding it.
 *   DONT_BREAK_LINES: don’t break lines at 76 characters
 *     &lt;i&gt;Note: Technically, this makes your encoding non-compliant.&lt;/i&gt;
 * </pre>
 *
 * <p>
 * Example: <code>encodeBytes( myData, Base64.GZIP )</code> or
 * <p>
 * Example:
 * <code>encodeBytes( myData, Base64.GZIP | Base64.DONT_BREAK_LINES )</code>
 *
 *
```

```
 * @param source
 *            The data to convert
 * @param options
 *            Specified options
 * @see Base64#GZIP
 * @see Base64#DONT_BREAK_LINES
 * @since 2.0
 */
public static String encodeBytes(byte[] source, int options) {
return encodeBytes(source, 0, source.length, options);
} // end encodeBytes

/**
 * Encodes a byte array into Base64 notation. Does not GZip-compress data.
 *
 * @param source
 *            The data to convert
 * @param off
 *            Offset in array where conversion should begin
 * @param len
 *            Length of data to convert
 * @since 1.4
 */
public static String encodeBytes(byte[] source, int off, int len) {
return encodeBytes(source, off, len, NO_OPTIONS);
} // end encodeBytes

/**
 * Encodes a byte array into Base64 notation.
 * <p>
 * Valid options:
 *
 * <pre>
 *   GZIP: gzip-compresses object before encoding it.
 *   DONT_BREAK_LINES: don’t break lines at 76 characters
 *     &lt;i&gt;Note: Technically, this makes your encoding non-compliant.&lt;/i&gt;
 * </pre>
 *
 * <p>
 * Example: <code>encodeBytes( myData, Base64.GZIP )</code> or
 * <p>
 * Example:
 * <code>encodeBytes( myData, Base64.GZIP | Base64.DONT_BREAK_LINES )</code>
 *
```

```
 *
 * @param source
 *            The data to convert
 * @param off
 *            Offset in array where conversion should begin
 * @param len
 *            Length of data to convert
 * @param options
 *            Specified options
 * @param options
 *            alphabet type is pulled from this (standard, url-safe,
 *            ordered)
 * @see Base64#GZIP
 * @see Base64#DONT_BREAK_LINES
 * @since 2.0
 */
public static String encodeBytes(byte[] source, int off, int len,
int options) {
// Isolate options
int dontBreakLines = (options & DONT_BREAK_LINES);
int gzip = (options & GZIP);

// Compress?
if (gzip == GZIP) {
java.io.ByteArrayOutputStream baos = null;
java.util.zip.GZIPOutputStream gzos = null;
Base64.OutputStream b64os = null;

try {
// GZip -> Base64 -> ByteArray
baos = new java.io.ByteArrayOutputStream();
b64os = new Base64.OutputStream(baos, ENCODE | options);
gzos = new java.util.zip.GZIPOutputStream(b64os);

gzos.write(source, off, len);
gzos.close();
} // end try
catch (java.io.IOException e) {
e.printStackTrace();
return null;
} // end catch
finally {
try {
gzos.close();
```

```
} catch (Exception e) {
}
try {
b64os.close();
} catch (Exception e) {
}
try {
baos.close();
} catch (Exception e) {
}
} // end finally

// Return value according to relevant encoding.
try {
return new String(baos.toByteArray(), PREFERRED_ENCODING);
} // end try
catch (java.io.UnsupportedEncodingException uue) {
return new String(baos.toByteArray());
} // end catch
} // end if: compress

// Else, don’t compress. Better not to use streams at all then.
else {
// Convert option to boolean in way that code likes it.
boolean breakLines = dontBreakLines == 0;

int len43 = len * 4 / 3;
byte[] outBuff = new byte[(len43) // Main 4:3
+ ((len % 3) > 0 ? 4 : 0) // Account for padding
+ (breakLines ? (len43 / MAX_LINE_LENGTH) : 0)]; // New
// lines
int d = 0;
int e = 0;
int len2 = len - 2;
int lineLength = 0;
for (; d < len2; d += 3, e += 4) {
encode3to4(source, d + off, 3, outBuff, e, options);

lineLength += 4;
if (breakLines && lineLength == MAX_LINE_LENGTH) {
outBuff[e + 4] = NEW_LINE;
e++;
lineLength = 0;
} // end if: end of line
```

```
} // en dfor: each piece of array

if (d < len) {
encode3to4(source, d + off, len - d, outBuff, e, options);
e += 4;
} // end if: some padding needed

// Return value according to relevant encoding.
try {
return new String(outBuff, 0, e, PREFERRED_ENCODING);
} // end try
catch (java.io.UnsupportedEncodingException uue) {
return new String(outBuff, 0, e);
} // end catch

} // end else: don’t compress

} // end encodeBytes

/* ******** D E C O D I N G M E T H O D S ******** */

/**
 * Decodes four bytes from array <var>source</var> and writes the resulting
 * bytes (up to three of them) to <var>destination</var>. The source and
 * destination arrays can be manipulated anywhere along their length by
 * specifying <var>srcOffset</var> and <var>destOffset</var>. This method
 * does not check to make sure your arrays are large enough to accomodate
 * <var>srcOffset</var> + 4 for the <var>source</var> array or
 * <var>destOffset</var> + 3 for the <var>destination</var> array. This
 * method returns the actual number of bytes that were converted from the
 * Base64 encoding.
 * <p>
 * This is the lowest level of the decoding methods with all possible
 * parameters.
 * </p>
 *
 *
 * @param source
 *            the array to convert
 * @param srcOffset
 *            the index where conversion begins
 * @param destination
 *            the array to hold the conversion
 * @param destOffset
```

```
 *            the index where output will be put
 * @param options
 *            alphabet type is pulled from this (standard, url-safe,
 *            ordered)
 * @return the number of decoded bytes converted
 * @since 1.3
 */
private static int decode4to3(byte[] source, int srcOffset,
byte[] destination, int destOffset, int options) {
byte[] DECODABET = getDecodabet(options);

// Example: Dk==
if (source[srcOffset + 2] == EQUALS_SIGN) {
// Two ways to do the same thing. Don’t know which way I like best.
// int outBuff = ( ( DECODABET[ source[ srcOffset ] ] << 24 ) >>> 6
// )
// | ( ( DECODABET[ source[ srcOffset + 1] ] << 24 ) >>> 12 );
int outBuff = ((DECODABET[source[srcOffset]] & 0xFF) << 18)
| ((DECODABET[source[srcOffset + 1]] & 0xFF) << 12);

destination[destOffset] = (byte) (outBuff >>> 16);
return 1;
}

// Example: DkL=
else if (source[srcOffset + 3] == EQUALS_SIGN) {
// Two ways to do the same thing. Don’t know which way I like best.
// int outBuff = ( ( DECODABET[ source[ srcOffset ] ] << 24 ) >>> 6
// )
// | ( ( DECODABET[ source[ srcOffset + 1 ] ] << 24 ) >>> 12 )
// | ( ( DECODABET[ source[ srcOffset + 2 ] ] << 24 ) >>> 18 );
int outBuff = ((DECODABET[source[srcOffset]] & 0xFF) << 18)
| ((DECODABET[source[srcOffset + 1]] & 0xFF) << 12)
| ((DECODABET[source[srcOffset + 2]] & 0xFF) << 6);

destination[destOffset] = (byte) (outBuff >>> 16);
destination[destOffset + 1] = (byte) (outBuff >>> 8);
return 2;
}

// Example: DkLE
else {
try {
// Two ways to do the same thing. Don’t know which way I like
```

```
// best.
// int outBuff = ( ( DECODABET[ source[ srcOffset ] ] << 24 )
// >>> 6 )
// | ( ( DECODABET[ source[ srcOffset + 1 ] ] << 24 ) >>> 12 )
// | ( ( DECODABET[ source[ srcOffset + 2 ] ] << 24 ) >>> 18 )
// | ( ( DECODABET[ source[ srcOffset + 3 ] ] << 24 ) >>> 24 );
int outBuff = ((DECODABET[source[srcOffset]] & 0xFF) << 18)
| ((DECODABET[source[srcOffset + 1]] & 0xFF) << 12)
| ((DECODABET[source[srcOffset + 2]] & 0xFF) << 6)
| ((DECODABET[source[srcOffset + 3]] & 0xFF));

destination[destOffset] = (byte) (outBuff >> 16);
destination[destOffset + 1] = (byte) (outBuff >> 8);
destination[destOffset + 2] = (byte) (outBuff);

return 3;
} catch (Exception e) {
System.out.println("" + source[srcOffset] + ": "
+ (DECODABET[source[srcOffset]]));
System.out.println("" + source[srcOffset + 1] + ": "
+ (DECODABET[source[srcOffset + 1]]));
System.out.println("" + source[srcOffset + 2] + ": "
+ (DECODABET[source[srcOffset + 2]]));
System.out.println("" + source[srcOffset + 3] + ": "
+ (DECODABET[source[srcOffset + 3]]));
return -1;
} // end catch
}
} // end decodeToBytes

/**
 * Very low-level access to decoding ASCII characters in the form of a byte
 * array. Does not support automatically gunzipping or any other "fancy"
 * features.
 *
 * @param source
 *            The Base64 encoded data
 * @param off
 *            The offset of where to begin decoding
 * @param len
 *            The length of characters to decode
 * @return decoded data
 * @since 1.3
 */
```

```
public static byte[] decode(byte[] source, int off, int len, int options) {
byte[] DECODABET = getDecodabet(options);

int len34 = len * 3 / 4;
byte[] outBuff = new byte[len34]; // Upper limit on size of output
int outBuffPosn = 0;

byte[] b4 = new byte[4];
int b4Posn = 0;
int i = 0;
byte sbiCrop = 0;
byte sbiDecode = 0;
for (i = off; i < off + len; i++) {
sbiCrop = (byte) (source[i] & 0x7f); // Only the low seven bits
sbiDecode = DECODABET[sbiCrop];

if (sbiDecode >= WHITE_SPACE_ENC) // White space, Equals sign or
// better
{
if (sbiDecode >= EQUALS_SIGN_ENC) {
b4[b4Posn++] = sbiCrop;
if (b4Posn > 3) {
outBuffPosn += decode4to3(b4, 0, outBuff, outBuffPosn,
options);
b4Posn = 0;

// If that was the equals sign, break out of ’for’ loop
if (sbiCrop == EQUALS_SIGN)
break;
} // end if: quartet built

} // end if: equals sign or better

} // end if: white space, equals sign or better
else {
System.err.println("Bad Base64 input character at " + i + ": "
+ source[i] + "(decimal)");
return null;
} // end else:
} // each input character

byte[] out = new byte[outBuffPosn];
System.arraycopy(outBuff, 0, out, 0, outBuffPosn);
return out;
```

```
} // end decode

/**
 * Decodes data from Base64 notation, automatically detecting
 * gzip-compressed data and decompressing it.
 *
 * @param s
 *            the string to decode
 * @return the decoded data
 * @since 1.4
 */
public static byte[] decode(String s) {
return decode(s, NO_OPTIONS);
}

/**
 * Decodes data from Base64 notation, automatically detecting
 * gzip-compressed data and decompressing it.
 *
 * @param s
 *            the string to decode
 * @param options
 *            encode options such as URL_SAFE
 * @return the decoded data
 * @since 1.4
 */
public static byte[] decode(String s, int options) {
byte[] bytes;
try {
bytes = s.getBytes(PREFERRED_ENCODING);
} // end try
catch (java.io.UnsupportedEncodingException uee) {
bytes = s.getBytes();
} // end catch
// </change>

// Decode
bytes = decode(bytes, 0, bytes.length, options);

// Check to see if it’s gzip-compressed
// GZIP Magic Two-Byte Number: 0x8b1f (35615)
if (bytes != null && bytes.length >= 4) {

int head = ((int) bytes[0] & 0xff) | ((bytes[1] << 8) & 0xff00);
```

```
if (java.util.zip.GZIPInputStream.GZIP_MAGIC == head) {
java.io.ByteArrayInputStream bais = null;
java.util.zip.GZIPInputStream gzis = null;
java.io.ByteArrayOutputStream baos = null;
byte[] buffer = new byte[2048];
int length = 0;

try {
baos = new java.io.ByteArrayOutputStream();
bais = new java.io.ByteArrayInputStream(bytes);
gzis = new java.util.zip.GZIPInputStream(bais);

while ((length = gzis.read(buffer)) >= 0) {
baos.write(buffer, 0, length);
} // end while: reading input

// No error? Get new bytes.
bytes = baos.toByteArray();

} // end try
catch (java.io.IOException e) {
// Just return originally-decoded bytes
} // end catch
finally {
try {
baos.close();
} catch (Exception e) {
}
try {
gzis.close();
} catch (Exception e) {
}
try {
bais.close();
} catch (Exception e) {
}
} // end finally

} // end if: gzipped
} // end if: bytes.length >= 2

return bytes;
} // end decode
```

```
/**
 * Attempts to decode Base64 data and deserialize a Java Object within.
 * Returns <tt>null</tt> if there was an error.
 *
 * @param encodedObject
 *            The Base64 data to decode
 * @return The decoded and deserialized object
 * @since 1.5
 */
public static Object decodeToObject(String encodedObject) {
// Decode and gunzip if necessary
byte[] objBytes = decode(encodedObject);

java.io.ByteArrayInputStream bais = null;
java.io.ObjectInputStream ois = null;
Object obj = null;

try {
bais = new java.io.ByteArrayInputStream(objBytes);
ois = new java.io.ObjectInputStream(bais);

obj = ois.readObject();
} // end try
catch (java.io.IOException e) {
e.printStackTrace();
obj = null;
} // end catch
catch (java.lang.ClassNotFoundException e) {
e.printStackTrace();
obj = null;
} // end catch
finally {
try {
bais.close();
} catch (Exception e) {
}
try {
ois.close();
} catch (Exception e) {
}
} // end finally

return obj;
} // end decodeObject
```

```
/**
 * Convenience method for encoding data to a file.
 *
 * @param dataToEncode
 *            byte array of data to encode in base64 form
 * @param filename
 *            Filename for saving encoded data
 * @return <tt>true</tt> if successful, <tt>false</tt> otherwise
 *
 * @since 2.1
 */
public static boolean encodeToFile(byte[] dataToEncode, String filename) {
boolean success = false;
Base64.OutputStream bos = null;
try {
bos = new Base64.OutputStream(
new java.io.FileOutputStream(filename), Base64.ENCODE);
bos.write(dataToEncode);
success = true;
} // end try
catch (java.io.IOException e) {

success = false;
} // end catch: IOException
finally {
try {
bos.close();
} catch (Exception e) {
}
} // end finally

return success;
} // end encodeToFile

/**
 * Convenience method for decoding data to a file.
 *
 * @param dataToDecode
 *            Base64-encoded data as a string
 * @param filename
 *            Filename for saving decoded data
 * @return <tt>true</tt> if successful, <tt>false</tt> otherwise
 *
```

```
 * @since 2.1
 */
public static boolean decodeToFile(String dataToDecode, String filename) {
boolean success = false;
Base64.OutputStream bos = null;
try {
bos = new Base64.OutputStream(
new java.io.FileOutputStream(filename), Base64.DECODE);
bos.write(dataToDecode.getBytes(PREFERRED_ENCODING));
success = true;
} // end try
catch (java.io.IOException e) {
success = false;
} // end catch: IOException
finally {
try {
bos.close();
} catch (Exception e) {
}
} // end finally

return success;
} // end decodeToFile

/**
 * Convenience method for reading a base64-encoded file and decoding it.
 *
 * @param filename
 *            Filename for reading encoded data
 * @return decoded byte array or null if unsuccessful
 *
 * @since 2.1
 */
public static byte[] decodeFromFile(String filename) {
byte[] decodedData = null;
Base64.InputStream bis = null;
try {
// Set up some useful variables
java.io.File file = new java.io.File(filename);
byte[] buffer = null;
int length = 0;
int numBytes = 0;

// Check for size of file
```

```
if (file.length() > Integer.MAX_VALUE) {
System.err
.println("File is too big for this convenience method ("
+ file.length() + " bytes).");
return null;
} // end if: file too big for int index
buffer = new byte[(int) file.length()];

// Open a stream
bis = new Base64.InputStream(new java.io.BufferedInputStream(
new java.io.FileInputStream(file)), Base64.DECODE);

// Read until done
while ((numBytes = bis.read(buffer, length, 4096)) >= 0)
length += numBytes;

// Save in a variable to return
decodedData = new byte[length];
System.arraycopy(buffer, 0, decodedData, 0, length);

} // end try
catch (java.io.IOException e) {
System.err.println("Error decoding from file " + filename);
} // end catch: IOException
finally {
try {
bis.close();
} catch (Exception e) {
}
} // end finally

return decodedData;
} // end decodeFromFile

/**
 * Convenience method for reading a binary file and base64-encoding it.
 *
 * @param filename
 *            Filename for reading binary data
 * @return base64-encoded string or null if unsuccessful
 *
 * @since 2.1
 */
public static String encodeFromFile(String filename) {
```

```
String encodedData = null;
Base64.InputStream bis = null;
try {
// Set up some useful variables
java.io.File file = new java.io.File(filename);
byte[] buffer = new byte[Math.max((int) (file.length() * 1.4), 40)]; // Need
// max()
// for
// math
// on
// small
// files
// (v2.2.1)
int length = 0;
int numBytes = 0;

// Open a stream
bis = new Base64.InputStream(new java.io.BufferedInputStream(
new java.io.FileInputStream(file)), Base64.ENCODE);

// Read until done
while ((numBytes = bis.read(buffer, length, 4096)) >= 0)
length += numBytes;

// Save in a variable to return
encodedData = new String(buffer, 0, length,
Base64.PREFERRED_ENCODING);

} // end try
catch (java.io.IOException e) {
System.err.println("Error encoding from file " + filename);
} // end catch: IOException
finally {
try {
bis.close();
} catch (Exception e) {
}
} // end finally

return encodedData;
} // end encodeFromFile

/**
 * Reads <tt>infile</tt> and encodes it to <tt>outfile</tt>.
```

```
 *
 * @param infile
 *            Input file
 * @param outfile
 *            Output file
 * @since 2.2
 */
public static void encodeFileToFile(String infile, String outfile) {
String encoded = Base64.encodeFromFile(infile);
java.io.OutputStream out = null;
try {
out = new java.io.BufferedOutputStream(
new java.io.FileOutputStream(outfile));
out.write(encoded.getBytes("US-ASCII")); // Strict, 7-bit output.
} // end try
catch (java.io.IOException ex) {
ex.printStackTrace();
} // end catch
finally {
try {
out.close();
} catch (Exception ex) {
}
} // end finally
} // end encodeFileToFile

/**
 * Reads <tt>infile</tt> and decodes it to <tt>outfile</tt>.
 *
 * @param infile
 *            Input file
 * @param outfile
 *            Output file
 * @since 2.2
 */
public static void decodeFileToFile(String infile, String outfile) {
byte[] decoded = Base64.decodeFromFile(infile);
java.io.OutputStream out = null;
try {
out = new java.io.BufferedOutputStream(
new java.io.FileOutputStream(outfile));
out.write(decoded);
} // end try
catch (java.io.IOException ex) {
```

```
ex.printStackTrace();
} // end catch
finally {
try {
out.close();
} catch (Exception ex) {
}
} // end finally
} // end decodeFileToFile

/* ******** I N N E R C L A S S I N P U T S T R E A M ******** */

/**
 * A {@link Base64.InputStream} will read data from another
 * <tt>java.io.InputStream</tt>, given in the constructor, and
 * encode/decode to/from Base64 notation on the fly.
 *
 * @see Base64
 * @since 1.3
 */
public static class InputStream extends java.io.FilterInputStream {
private boolean encode; // Encoding or decoding

private int position; // Current position in the buffer

private byte[] buffer; // Small buffer holding converted data

private int bufferLength; // Length of buffer (3 or 4)

private int numSigBytes; // Number of meaningful bytes in the buffer

private int lineLength;

private boolean breakLines; // Break lines at less than 80 characters

private int options; // Record options used to create the stream.

private byte[] alphabet; // Local copies to avoid extra method calls

private byte[] decodabet; // Local copies to avoid extra method calls

/**
 * Constructs a {@link Base64.InputStream} in DECODE mode.
 *
```

```
 * @param in
 *            the <tt>java.io.InputStream</tt> from which to read
 *            data.
 * @since 1.3
 */
public InputStream(java.io.InputStream in) {
this(in, DECODE);
} // end constructor

/**
 * Constructs a {@link Base64.InputStream} in either ENCODE or DECODE
 * mode.
 * <p>
 * Valid options:
 *
 * <pre>
 *   ENCODE or DECODE: Encode or Decode as data is read.
 *   DONT_BREAK_LINES: don't break lines at 76 characters
 *     (only meaningful when encoding)
 *     &lt;i&gt;Note: Technically, this makes your encoding non-compliant.&lt;/i&gt;
 * </pre>
 *
 * <p>
 * Example: <code>new Base64.InputStream( in, Base64.DECODE )</code>
 *
 *
 * @param in
 *            the <tt>java.io.InputStream</tt> from which to read
 *            data.
 * @param options
 *            Specified options
 * @see Base64#ENCODE
 * @see Base64#DECODE
 * @see Base64#DONT_BREAK_LINES
 * @since 2.0
 */
public InputStream(java.io.InputStream in, int options) {
super(in);
this.breakLines = (options & DONT_BREAK_LINES) != DONT_BREAK_LINES;
this.encode = (options & ENCODE) == ENCODE;
this.bufferLength = encode ? 4 : 3;
this.buffer = new byte[bufferLength];
this.position = -1;
this.lineLength = 0;
```

```
this.options = options; // Record for later, mostly to determine
// which alphabet to use
this.alphabet = getAlphabet(options);
this.decodabet = getDecodabet(options);
} // end constructor

/**
 * Reads enough of the input stream to convert to/from Base64 and
 * returns the next byte.
 *
 * @return next byte
 * @since 1.3
 */
public int read() throws java.io.IOException {
// Do we need to get data?
if (position < 0) {
if (encode) {
byte[] b3 = new byte[3];
int numBinaryBytes = 0;
for (int i = 0; i < 3; i++) {
try {
int b = in.read();

// If end of stream, b is -1.
if (b >= 0) {
b3[i] = (byte) b;
numBinaryBytes++;
} // end if: not end of stream

} // end try: read
catch (java.io.IOException e) {
// Only a problem if we got no data at all.
if (i == 0)
throw e;

} // end catch
} // end for: each needed input byte

if (numBinaryBytes > 0) {
encode3to4(b3, 0, numBinaryBytes, buffer, 0, options);
position = 0;
numSigBytes = 4;
} // end if: got data
else {
```

```
return -1;
} // end else
} // end if: encoding

// Else decoding
else {
byte[] b4 = new byte[4];
int i = 0;
for (i = 0; i < 4; i++) {
// Read four "meaningful" bytes:
int b = 0;
do {
b = in.read();
} while (b >= 0
&& decodabet[b & 0x7f] <= WHITE_SPACE_ENC);

if (b < 0)
break; // Reads a -1 if end of stream

b4[i] = (byte) b;
} // end for: each needed input byte

if (i == 4) {
numSigBytes = decode4to3(b4, 0, buffer, 0, options);
position = 0;
} // end if: got four characters
else if (i == 0) {
return -1;
} // end else if: also padded correctly
else {
// Must have broken out from above.
throw new java.io.IOException(
"Improperly padded Base64 input.");
} // end

} // end else: decode
} // end else: get data

// Got data?
if (position >= 0) {
// End of relevant data?
if ( /* !encode && */position >= numSigBytes)
return -1;
```

```
if (encode && breakLines && lineLength >= MAX_LINE_LENGTH) {
lineLength = 0;
return ’\n’;
} // end if
else {
lineLength++; // This isn’t important when decoding
// but throwing an extra "if" seems
// just as wasteful.

int b = buffer[position++];

if (position >= bufferLength)
position = -1;

return b & 0xFF; // This is how you "cast" a byte that’s
// intended to be unsigned.
} // end else
} // end if: position >= 0

// Else error
else {
// When JDK1.4 is more accepted, use an assertion here.
throw new java.io.IOException(
"Error in Base64 code reading stream.");
} // end else
} // end read

/**
 * Calls {@link #read()} repeatedly until the end of stream is reached
 * or <var>len</var> bytes are read. Returns number of bytes read into
 * array or -1 if end of stream is encountered.
 *
 * @param dest
 *            array to hold values
 * @param off
 *            offset for array
 * @param len
 *            max number of bytes to read into array
 * @return bytes read into array or -1 if end of stream is encountered.
 * @since 1.3
 */
public int read(byte[] dest, int off, int len)
throws java.io.IOException {
int i;
```

```
int b;
for (i = 0; i < len; i++) {
b = read();

// if( b < 0 && i == 0 )
// return -1;

if (b >= 0)
dest[off + i] = (byte) b;
else if (i == 0)
return -1;
else
break; // Out of ’for’ loop
} // end for: each byte read
return i;
} // end read

} // end inner class InputStream

/* ******** I N N E R C L A S S O U T P U T S T R E A M ******** */

/**
 * A {@link Base64.OutputStream} will write data to another
 * <tt>java.io.OutputStream</tt>, given in the constructor, and
 * encode/decode to/from Base64 notation on the fly.
 *
 * @see Base64
 * @since 1.3
 */
public static class OutputStream extends java.io.FilterOutputStream {
private boolean encode;

private int position;

private byte[] buffer;

private int bufferLength;

private int lineLength;

private boolean breakLines;

private byte[] b4; // Scratch used in a few places
```

```
private boolean suspendEncoding;

private int options; // Record for later

private byte[] alphabet; // Local copies to avoid extra method calls

private byte[] decodabet; // Local copies to avoid extra method calls

/**
 * Constructs a {@link Base64.OutputStream} in ENCODE mode.
 *
 * @param out
 *            the <tt>java.io.OutputStream</tt> to which data will be
 *            written.
 * @since 1.3
 */
public OutputStream(java.io.OutputStream out) {
this(out, ENCODE);
} // end constructor

/**
 * Constructs a {@link Base64.OutputStream} in either ENCODE or DECODE
 * mode.
 * <p>
 * Valid options:
 *
 * <pre>
 *   ENCODE or DECODE: Encode or Decode as data is read.
 *   DONT_BREAK_LINES: don’t break lines at 76 characters
 *     (only meaningful when encoding)
 *     &lt;i&gt;Note: Technically, this makes your encoding non-compliant.&lt;/i&gt;
 * </pre>
 *
 * <p>
 * Example: <code>new Base64.OutputStream( out, Base64.ENCODE )</code>
 *
 * @param out
 *            the <tt>java.io.OutputStream</tt> to which data will be
 *            written.
 * @param options
 *            Specified options.
 * @see Base64#ENCODE
 * @see Base64#DECODE
 * @see Base64#DONT_BREAK_LINES
```

```
 * @since 1.3
 */
public OutputStream(java.io.OutputStream out, int options) {
super(out);
this.breakLines = (options & DONT_BREAK_LINES) != DONT_BREAK_LINES;
this.encode = (options & ENCODE) == ENCODE;
this.bufferLength = encode ? 3 : 4;
this.buffer = new byte[bufferLength];
this.position = 0;
this.lineLength = 0;
this.suspendEncoding = false;
this.b4 = new byte[4];
this.options = options;
this.alphabet = getAlphabet(options);
this.decodabet = getDecodabet(options);
} // end constructor

/**
 * Writes the byte to the output stream after converting to/from Base64
 * notation. When encoding, bytes are buffered three at a time before
 * the output stream actually gets a write() call. When decoding, bytes
 * are buffered four at a time.
 *
 * @param theByte
 *            the byte to write
 * @since 1.3
 */
public void write(int theByte) throws java.io.IOException {
// Encoding suspended?
if (suspendEncoding) {
super.out.write(theByte);
return;
} // end if: supsended

// Encode?
if (encode) {
buffer[position++] = (byte) theByte;
if (position >= bufferLength) // Enough to encode.
{
out.write(encode3to4(b4, buffer, bufferLength, options));

lineLength += 4;
if (breakLines && lineLength >= MAX_LINE_LENGTH) {
out.write(NEW_LINE);
```

```
lineLength = 0;
} // end if: end of line

position = 0;
} // end if: enough to output
} // end if: encoding

// Else, Decoding
else {
// Meaningful Base64 character?
if (decodabet[theByte & 0x7f] > WHITE_SPACE_ENC) {
buffer[position++] = (byte) theByte;
if (position >= bufferLength) // Enough to output.
{
int len = Base64.decode4to3(buffer, 0, b4, 0, options);
out.write(b4, 0, len);
// out.write( Base64.decode4to3( buffer ) );
position = 0;
} // end if: enough to output
} // end if: meaningful base64 character
else if (decodabet[theByte & 0x7f] != WHITE_SPACE_ENC) {
throw new java.io.IOException(
"Invalid character in Base64 data.");
} // end else: not white space either
} // end else: decoding
} // end write

/**
 * Calls {@link #write(int)} repeatedly until <var>len</var> bytes are
 * written.
 *
 * @param theBytes
 *            array from which to read bytes
 * @param off
 *            offset for array
 * @param len
 *            max number of bytes to read into array
 * @since 1.3
 */
public void write(byte[] theBytes, int off, int len)
throws java.io.IOException {
// Encoding suspended?
if (suspendEncoding) {
super.out.write(theBytes, off, len);
```

```
return;
} // end if: supsended

for (int i = 0; i < len; i++) {
write(theBytes[off + i]);
} // end for: each byte written

} // end write

/**
 * Method added by PHIL. [Thanks, PHIL. -Rob] This pads the buffer
 * without closing the stream.
 */
public void flushBase64() throws java.io.IOException {
if (position > 0) {
if (encode) {
out.write(encode3to4(b4, buffer, position, options));
position = 0;
} // end if: encoding
else {
throw new java.io.IOException(
"Base64 input not properly padded.");
} // end else: decoding
} // end if: buffer partially full

} // end flush

/**
 * Flushes and closes (I think, in the superclass) the stream.
 *
 * @since 1.3
 */
public void close() throws java.io.IOException {
// 1. Ensure that pending characters are written
flushBase64();

// 2. Actually close the stream
// Base class both flushes and closes.
super.close();

buffer = null;
out = null;
} // end close
```

```
/**
 * Suspends encoding of the stream. May be helpful if you need to embed
 * a piece of base640-encoded data in a stream.
 *
 * @since 1.5.1
 */
public void suspendEncoding() throws java.io.IOException {
flushBase64();
this.suspendEncoding = true;
} // end suspendEncoding

/**
 * Resumes encoding of the stream. May be helpful if you need to embed a
 * piece of base640-encoded data in a stream.
 *
 * @since 1.5.1
 */
public void resumeEncoding() {
this.suspendEncoding = false;
} // end resumeEncoding

} // end inner class OutputStream

} // end class Base64
package br.ufsc.jpki.test;

import java.security.cert.X509Certificate;
import java.util.Calendar;

import br.ufsc.jpki.ca.CertificationAuthority;
import br.ufsc.jpki.ca.CertificationAuthorityBuilder;
import br.ufsc.jpki.container.CertParameters;
import br.ufsc.jpki.io.FileHandler;
import br.ufsc.jpki.persistence.HCertificateDAO;
import br.ufsc.jpki.persistence.HCertificationAuthorityDAO;
import br.ufsc.jpki.persistence.HibernateUtil;
import br.ufsc.jpki.persistence.dao.CertificateDAO;
import br.ufsc.jpki.persistence.dao.CertificationAuthorityDAO;
import br.ufsc.jpki.provider.CryptographicServicesProvider;

import junit.framework.TestCase;

public class CertificateDAOTest extends TestCase {
```

```
private X509Certificate certificate = null;

protected void setUp() throws Exception {
super.setUp();
this.certificate = FileHandler
.loadCertificate("/home/jeanms/temp/certificadoACRaiz.crt");
}

public void tearDown() {
HibernateUtil.getSessionFactory().close();
}

public void testStoreAndLoadCertificate() {
CertificationAuthorityDAO caDAO = new HCertificationAuthorityDAO();
CertificationAuthority loadedCa = null;
try {
loadedCa = caDAO.loadCA("AC-LabSEC");
} catch (Exception ex) {
fail(ex.getMessage());
}
assertNotNull(loadedCa);
CertificateDAO dao = new HCertificateDAO();
X509Certificate cert = null;
try {
dao.storeCertificate(this.certificate, loadedCa.getLabel());
cert = dao.loadCertificate(this.certificate.getSerialNumber(),
loadedCa.getLabel());
} catch (Exception e) {
fail(e.getMessage());
}
assertNotNull(cert);
assertEquals(cert, this.certificate);
HibernateUtil.getSessionFactory().close();
}

}
package br.ufsc.jpki.test;

import java.security.KeyPair;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;

import br.ufsc.jpki.provider.CryptographicServicesProvider;
import br.ufsc.jpki.provider.CryptographicServicesProvider.KeyType;
```

```
import junit.framework.TestCase;

public class CryptographicServiceProviderTest extends TestCase {

/*
 * Test method for
 * 'br.ufsc.jpki.provider.CryptographicServiceProvider.generateKeyPair(KeyType)'
 */
public void testGenerateKeyPair() {
CryptographicServicesProvider provider = new CryptographicServicesProvider();
KeyPair kp = null;
try {
kp = provider
.generateKeyPair(CryptographicServicesProvider.KeyType.RSA);
} catch (NoSuchAlgorithmException e) {
fail(e.getMessage());
}
CryptographicServiceProviderTest
.assertNotNull("keyPair RSA = null", kp);
try {
kp = provider
.generateKeyPair(CryptographicServicesProvider.KeyType.DSA);
} catch (NoSuchAlgorithmException e) {
fail(e.getMessage());
}
CryptographicServiceProviderTest
.assertNotNull("keyPair DSA = null", kp);
}

/*
 * Test method for
 * 'br.ufsc.jpki.provider.CryptographicServiceProvider.generateRandom()'
 */
public void testGenerateRandom() {
CryptographicServicesProvider provider = new CryptographicServicesProvider();
SecureRandom rand = provider.generateRandom();
CryptographicServiceProviderTest.assertNotNull(rand);
}

}
package br.ufsc.jpki.test;

import java.util.Calendar;
```

```
import br.ufsc.jpki.ca.CertificationAuthorityBuilder;
import br.ufsc.jpki.container.CertParameters;
import br.ufsc.jpki.provider.CryptographicServicesProvider;
import junit.framework.TestCase;

public class CertificationAuthorityBuilderTest extends TestCase {

/*
 * Test method for 'br.ufsc.jpki.CertificationAuthority.getNextSerial()'
 */
public void testBuildCA() {
CryptographicServicesProvider provider = new CryptographicServicesProvider();
CertificationAuthorityBuilder caBuilder = new CertificationAuthorityBuilder(
provider);
CertParameters cp = new CertParameters();
cp.setCommonName("AC-LabSEC");
Calendar cal = Calendar.getInstance();
cp.setNotBefore(cal);
cal.add(Calendar.YEAR, 10);
cp.setNotAfter(cal);

caBuilder.setLabel("AC-LabSEC");
caBuilder.setCertParameters(cp);
}
}
package br.ufsc.jpki.test;

import java.math.BigInteger;
import java.security.Security;
import java.security.cert.X509Certificate;
import java.security.interfaces.RSAPrivateKey;
import java.util.Calendar;
import java.util.Collection;
import java.util.Enumeration;

import org.bouncycastle.jce.provider.BouncyCastleProvider;
import org.bouncycastle.x509.X509V3CertificateGenerator;

import br.ufsc.jpki.PKIManager;
import br.ufsc.jpki.io.FileHandler;
import br.ufsc.jpki.provider.PKCS11Handler;
import junit.framework.TestCase;
```

```
public class PKCS11HandlerTest extends TestCase {

private PKCS11Handler pkcs11;

public void setUp() {
try {
this.pkcs11 = new PKCS11Handler();
} catch (Exception e) {
fail("Failure in constructor: " + e.getMessage());
}
}

public void testGeneratePrivateKey() {
try {
this.pkcs11.generatePrivateKey();
} catch (Exception e) {
fail(e.getMessage());
}
}

// public void testGetAvailableCertificates() {
// try {
// Enumeration<String> availableCerts =
// this.pkcs11.getAvailableCertificates();
// while (availableCerts.hasMoreElements()) {
// System.out.println(availableCerts.nextElement());
// }
// }
// catch(Exception e) {
// fail(e.getMessage());
// }
// }

// public void testLoadCertificate() {
// try {
// this.pkcs11.loadCertificate(this.pkcs11.getAvailableCertificates().nextElement());
// }
// catch(Exception e) {
// fail(e.getMessage());
// }
// }
//
// public void testProvePossession() {
// try {
```

```
// this.pkcs11.provePossession(this.pkcs11.getAvailableCertificates().nextElement());
// }
// catch(Exception e) {
// fail(e.getMessage());
// }
// }
//
// public void testStoreCertificate () {
// Security.addProvider(new BouncyCastleProvider());
// try {
// //String label = this.pkcs11.getAvailableCertificates().nextElement();
// X509Certificate cert =
// ASN1EncodedFileHandler.loadCertificate("/home/jeanms/Documents/tcc/referencias/ICP-Brasil/certificadoACRaiz
// this.pkcs11.storeCertificate("root", cert);
// }
// catch(Exception e) {
// e.printStackTrace();
// fail(e.getMessage());
// }
// }

}
package br.ufsc.jpki.test;

import java.util.Calendar;

import br.ufsc.jpki.ca.CertificationAuthority;
import br.ufsc.jpki.ca.CertificationAuthorityBuilder;
import br.ufsc.jpki.container.CertParameters;
import br.ufsc.jpki.io.FileHandler;
import br.ufsc.jpki.persistence.HCertificationAuthorityDAO;
import br.ufsc.jpki.persistence.HibernateUtil;
import br.ufsc.jpki.persistence.dao.CertificationAuthorityDAO;
import br.ufsc.jpki.provider.CryptographicServicesProvider;
import junit.framework.TestCase;

public class CertificationAuthorityDAOTest extends TestCase {

private CertificationAuthority ca;

protected void setUp() throws Exception {
CryptographicServicesProvider provider = new CryptographicServicesProvider();
CertificationAuthorityBuilder caBuilder = new CertificationAuthorityBuilder(
provider);
```

```
CertParameters cp = new CertParameters();
cp.setCommonName("AC-LabSEC");
Calendar cal = Calendar.getInstance();
cp.setNotBefore(cal);
cal.add(Calendar.YEAR, 10);
cp.setNotAfter(cal);

caBuilder.setLabel("AC-LabSEC");
caBuilder.setCertParameters(cp);
try {
this.ca = caBuilder.build();
} catch (Exception ex) {
fail(ex.getMessage());
}
HibernateUtil.cleanDatabase();
}

public void tearDown() {
HibernateUtil.getSessionFactory().close();
}

public void testStoreCA() {
CertificationAuthorityDAO caDAO = new HCertificationAuthorityDAO();
try {
// caDAO.storeCA(this.ca);
} catch (Exception ex) {
fail(ex.getMessage());
}
CertificationAuthority loadedCa = null;
try {
loadedCa = caDAO.loadCA("AC-LabSEC");
} catch (Exception ex) {
fail(ex.getMessage());
}
assertNotNull(loadedCa);
assertEquals(this.ca, loadedCa);
}

}
package br.ufsc.jpki.test;

import java.security.cert.X509Certificate;
import org.bouncycastle.asn1.pkcs.CertificationRequest;
```

```
import br.ufsc.jpki.io.FileHandler;
import junit.framework.TestCase;

public class ASN1EncodedFileLoaderTest extends TestCase {

/*
 * Test method for
 * 'br.ufsc.jpki.io.ASN1EncodedFileLoader.loadCertificate(String)'
 */
public void testLoadCertificate() {
X509Certificate certificate = null;
try {
certificate = FileHandler
.loadCertificate("/home/jeanms/temp/certificadoACRaiz.crt");
ASN1EncodedFileLoaderTest.assertNotNull("certificate == null",
certificate);
} catch (Exception ex) {
ASN1EncodedFileLoaderTest.fail(ex.getMessage());
}
}

/*
 * Test method for
 * 'br.ufsc.jpki.io.ASN1EncodedFileLoader.loadCertificateRequest(String)'
 */
public void testLoadCertificateRequest() {
CertificationRequest request = null;
try {
request = FileHandler
.loadCertificateRequest("/home/jeanms/temp/request.der");
ASN1EncodedFileLoaderTest.assertNotNull("request == null", request);
} catch (Exception ex) {
ASN1EncodedFileLoaderTest.fail(ex.getMessage());
}
}

}
package br.ufsc.jpki.test;

import java.security.cert.X509Certificate;
import java.util.Calendar;

import br.ufsc.jpki.ca.CertificationAuthority;
import br.ufsc.jpki.ca.CertificationAuthorityBuilder;
```

```
import br.ufsc.jpki.container.CertParameters;
import br.ufsc.jpki.io.FileHandler;
import br.ufsc.jpki.provider.CryptographicServicesProvider;
import junit.framework.TestCase;

public class CertificationAuthorityTest extends TestCase {

private CertificationAuthority ca;

public void setUp() {
CryptographicServicesProvider provider = new CryptographicServicesProvider();
CertificationAuthorityBuilder caBuilder = new CertificationAuthorityBuilder(
provider);
CertParameters cp = new CertParameters();
cp.setCommonName("AC-LabSEC");
Calendar cal = Calendar.getInstance();
cp.setNotBefore(cal);
cal.add(Calendar.YEAR, 10);
cp.setNotAfter(cal);

caBuilder.setLabel("AC-LabSEC");
caBuilder.setCertParameters(cp);
try {
this.ca = caBuilder.build();
} catch (Exception ex) {
fail(ex.getMessage());
}
}

/*
 * Test method for
 * 'br.ufsc.jpki.CertificationAuthority.issueCertificate(X509Certificate,
 * Calendar, Calendar)'
 */
public void testIssueCertificateX509CertificateCalendarCalendar() {
try {
X509Certificate oldCertificate = FileHandler
.loadCertificate("/home/jeanms/temp/certificadoACRaiz.crt");
Calendar notBefore = Calendar.getInstance();
Calendar notAfter = Calendar.getInstance();
notAfter.add(Calendar.YEAR, 10);
this.ca.renewCertificate(oldCertificate, notAfter, notBefore);
} catch (Exception ex) {
CertificationAuthorityBuilderTest.fail(ex.getMessage());
```

```
}
}

/*
 * Test method for
 * 'br.ufsc.jpki.CertificationAuthority.issueCertificate(X509Certificate,
 * Calendar)'
 */
public void testIssueCertificateX509CertificateCalendar() {
try {
X509Certificate certificate = FileHandler
.loadCertificate("/home/jeanms/temp/certificadoACRaiz.crt");
Calendar notAfter = Calendar.getInstance();
notAfter.add(Calendar.YEAR, 10);
this.ca.renewCertificate(certificate, notAfter);
} catch (Exception ex) {
CertificationAuthorityBuilderTest.fail(ex.getMessage());
}
}

/*
 * Test method for
 * 'br.ufsc.jpki.CertificationAuthority.issueCertificate(CertificationRequest)'
 */
public void testIssueCertificateCertificationRequest() {
fail("Not yet implemented");
}

}
package br.ufsc.jpki.persistence.dao;

import java.math.BigInteger;
import java.security.cert.Certificate;
import java.security.cert.X509Certificate;
import java.util.Calendar;
import java.util.Collection;
import java.util.Date;
import java.util.Iterator;
import java.util.List;
import java.util.Set;

import br.ufsc.jpki.ca.CertificationAuthority;
import br.ufsc.jpki.exception.DataLoadingException;
import br.ufsc.jpki.exception.DataWritingException;
```

```
import br.ufsc.jpki.persistence.RevokedCertificate;

public interface CertificateDAO {
public void storeCertificate(X509Certificate cert, String caLabel)
throws DataWritingException;

public X509Certificate loadCertificate(BigInteger serial, String caLabel)
throws DataLoadingException;

public X509Certificate loadCertificate(String serial, byte[] subjKeyId,
String caLabel) throws DataLoadingException;

public Collection<X509Certificate> searchForCertificates(String cnPattern,
String caLabel);

public Collection<X509Certificate> loadCertificatesFromIssueDate(
Calendar notBefore, Calendar notAfter, String caLabel);

public void storeRevokedCertificate(BigInteger certSerial, int reasonCode,
String caLabel) throws DataWritingException;

public List<RevokedCertificate> loadRevokedCertificates(Date notAfter,
String caLabel) throws DataLoadingException;
}
package br.ufsc.jpki.persistence.dao;

import java.math.BigInteger;
import java.security.cert.X509CRL;

import br.ufsc.jpki.exception.DataLoadingException;
import br.ufsc.jpki.exception.DataWritingException;

public interface CertificateRevocationListDAO {
public X509CRL loadCRL(BigInteger serial, String caLabel)
throws DataLoadingException;

public void storeCRL(X509CRL crl, String caLabel)
throws DataWritingException;
}
package br.ufsc.jpki.persistence.dao;

import java.math.BigInteger;
import java.util.Collection;
```

```
import org.bouncycastle.asn1.x509.SubjectKeyIdentifier;

import br.ufsc.jpki.ca.CertificationAuthority;
import br.ufsc.jpki.exception.DataLoadingException;
import br.ufsc.jpki.exception.DataWritingException;

public interface CertificationAuthorityDAO {
public CertificationAuthority loadCA(String label)
throws DataLoadingException;

public CertificationAuthority loadCA(byte[] keyId, BigInteger serial)
throws DataLoadingException;

public Collection<CertificationAuthority> loadSubordinatedCAs(String caLabel)
throws DataLoadingException;

public Collection<String> loadCAsLabels() throws DataLoadingException;

public Collection<CertificationAuthority> loadCAs()
throws DataLoadingException;

public void storeCA(CertificationAuthority ca,
CertificationAuthority issuingCA) throws DataWritingException;

public boolean exists(String caLabel) throws DataLoadingException;

public void updateCA(CertificationAuthority ca) throws DataWritingException;
}
package br.ufsc.jpki.persistence;

import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.math.BigInteger;
import java.security.cert.Certificate;
import java.security.cert.CertificateEncodingException;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Collection;
import java.util.Date;
import java.util.Iterator;
import java.util.LinkedList;
```

```
import java.util.List;
import java.util.Set;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import br.ufsc.jpki.ca.CertificationAuthority;
import br.ufsc.jpki.exception.DataLoadingException;
import br.ufsc.jpki.exception.DataWritingException;
import br.ufsc.jpki.persistence.dao.CertificateDAO;
import br.ufsc.jpki.persistence.dao.CertificationAuthorityDAO;
import br.ufsc.jpki.util.Base64;

public class HCertificateDAO implements CertificateDAO {

public X509Certificate loadCertificate(BigInteger serial, String caLabel)
throws DataLoadingException {
X509Certificate cert = null;
try {
Session session = HibernateUtil.getSessionFactory()
.getCurrentSession();
session.beginTransaction();
Query q = session
.createQuery("from PersistentCertificate as pc where pc.serial = "
+ serial
+ " and pc.issuer.label = '"
+ caLabel
+ "'");
PersistentCertificate pc = (PersistentCertificate) q.uniqueResult();
if (pc != null) {
cert = pc.toX509Certificate();
}
session.getTransaction().commit();
} catch (Exception e) {
throw new DataLoadingException("Error loading Certificate: "
+ e.getLocalizedMessage(), e);
}
return cert;
}

public void storeCertificate(X509Certificate cert, String caLabel)
throws DataWritingException {
PersistentCertificate pc = null;
```

```
PersistentCertificationAuthority pca = null;
try {
HCertificationAuthorityDAO caDao = new HCertificationAuthorityDAO();
pca = caDao.loadPCA(caLabel);
} catch (DataLoadingException e) {
throw new DataWritingException(
"Error loading the issuer certification auhtority: "
+ e.getLocalizedMessage(), e);
}
try {
pc = PersistentCertificate.fromX509Certificate(cert);
pc.setIssuer(pca);
Session session = HibernateUtil.getSessionFactory()
.getCurrentSession();
session.beginTransaction();
session.save(pc);
session.getTransaction().commit();
} catch (Exception e) {
throw new DataWritingException("Error storing certificate: "
+ e.getLocalizedMessage(), e);
}
}

public Collection<X509Certificate> loadCertificatesFromIssueDate(
Calendar notBefore, Calendar notAfter, String caLabel) {
// TODO Auto-generated method stub
return null;
}

public Collection<X509Certificate> searchForCertificates(String cnPattern,
String caLabel) {
// TODO Auto-generated method stub
return null;
}

public X509Certificate loadCertificate(String serial, byte[] subjKeyId,
String caLabel) throws DataLoadingException {
X509Certificate cert = null;
String serialClause = "";
if (serial != serial) {
serialClause = "pc.serial = " + serial;
}
try {
Session session = HibernateUtil.getSessionFactory()
```

```
.getCurrentSession();
session.beginTransaction();
Query q = session
.createQuery("from PersistentCertificate as pc where pc.issuer.label = '"
+ caLabel
+ "'"
+ " and pc.b64SubjKeyId = '"
+ Base64.encodeBytes(subjKeyId)
+ "'"
+ serialClause);
Iterator<PersistentCertificate> pcs = (Iterator<PersistentCertificate>) q
.iterate();
if (pcs != null && pcs.hasNext()) {
PersistentCertificate pc = pcs.next();
while (pcs.hasNext()) {
PersistentCertificate element = pcs.next();
if (element.getNotAfter().compareTo(pc.getNotAfter()) > 0) {
pc = element;
}
}
ByteArrayInputStream inStream = new ByteArrayInputStream(Base64
.decode(pc.getB64Encoding()));
CertificateFactory certFactory = CertificateFactory
.getInstance("X509");
cert = (X509Certificate) certFactory
.generateCertificate(inStream);
inStream.close();
}
session.getTransaction().commit();
} catch (Exception e) {
throw new DataLoadingException("Error loading Certificate: "
+ e.getLocalizedMessage(), e);
}
return cert;
}

public void storeRevokedCertificate(BigInteger certSerial, int reasonCode,
String caLabel) throws DataWritingException {
try {
Session session = HibernateUtil.getSessionFactory()
.getCurrentSession();
Transaction t = session.beginTransaction();
Query q = session
.createQuery("from PersistentCertificate as pc where pc.serial = "
```

```
+ certSerial
+ " and pc.issuer.label = '"
+ caLabel
+ "'");
RevokedCertificate rc = new RevokedCertificate();
rc.setCertificate((PersistentCertificate) q.uniqueResult());
rc.setReasonCode(reasonCode);
rc.setRevocationDate(Calendar.getInstance().getTime());
session.save(rc);
t.commit();
} catch (Exception e) {
throw new DataWritingException(
"Error storing revoked certificate: "
+ e.getLocalizedMessage(), e);
}
}

public List<RevokedCertificate> loadRevokedCertificates(Date notAfter,
String caLabel) throws DataLoadingException {
List<RevokedCertificate> rcs = new ArrayList<RevokedCertificate>();
try {
Session session = HibernateUtil.getSessionFactory()
.getCurrentSession();
session.beginTransaction();
Query q = session
.createQuery(
"from RevokedCertificate as rc where rc.certificate.issuer.label = ? and rc.certificate.notAfter > ? ")
.setString(0, caLabel).setDate(1, notAfter);
Iterator<RevokedCertificate> iter = q.iterate();
while (iter.hasNext()) {
RevokedCertificate rc = (RevokedCertificate) iter.next();
rc.getCertificate().getSerial();
rcs.add(rc);
}
session.getTransaction().commit();
} catch (Exception e) {
throw new DataLoadingException(
"Error storing revoked certificates: "
+ e.getLocalizedMessage(), e);
}
return rcs;
}

}
```

```
package br.ufsc.jpki.persistence;

import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.math.BigInteger;
import java.security.cert.CertificateEncodingException;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
import java.util.Date;

import org.bouncycastle.asn1.x509.SubjectKeyIdentifier;
import org.bouncycastle.asn1.x509.X509Extensions;
import org.bouncycastle.x509.extension.SubjectKeyIdentifierStructure;

import com.sun.org.apache.xpath.internal.operations.Equals;

import br.ufsc.jpki.ca.CertificationAuthority;
import br.ufsc.jpki.util.Base64;

public class PersistentCertificate {

protected Long id;

protected BigInteger serial;

protected String commonName;

protected String b64SubjKeyId;

protected String b64Encoding;

protected Date notAfter;

protected PersistentCertificationAuthority issuer;

public PersistentCertificate() {
super();
}

public String getCommonName() {
return commonName;
}
```

```
public void setCommonName(String commonName) {
this.commonName = commonName;
}

public BigInteger getSerial() {
return serial;
}

public void setSerial(BigInteger serial) {
this.serial = serial;
}

public static PersistentCertificate fromX509Certificate(
X509Certificate x509Cert) throws CertificateException {
PersistentCertificate pc = new PersistentCertificate();
pc.setSerial(x509Cert.getSerialNumber());
pc.setCommonName(x509Cert.getSubjectDN().getName());
pc.setNotAfter(x509Cert.getNotAfter());
byte[] binSubjKeyId = x509Cert
.getExtensionValue(X509Extensions.SubjectKeyIdentifier.getId());
if (binSubjKeyId == null) {
throw new CertificateException("No subject key identifier found.");
}
SubjectKeyIdentifierStructure subjKeyId = null;
try {
subjKeyId = new SubjectKeyIdentifierStructure(binSubjKeyId);
} catch (IOException e) {
throw new CertificateException("Invalid subject key identifier: "
+ e.getLocalizedMessage(), e);
}
pc.setB64SubjKeyId(Base64.encodeBytes(subjKeyId.getKeyIdentifier()));
pc.setB64Encoding(Base64.encodeBytes(x509Cert.getEncoded()));
return pc;
}

public X509Certificate toX509Certificate()
throws CertificateEncodingException, CertificateException,
IOException {
ByteArrayInputStream inStream = new ByteArrayInputStream(Base64
.decode(this.getB64Encoding()));
CertificateFactory certFactory = CertificateFactory.getInstance("X509");
X509Certificate cert = (X509Certificate) certFactory
.generateCertificate(inStream);
inStream.close();
```

```
return cert;
}

public boolean equals(Object obj) {
boolean equals = false;
if (obj instanceof PersistentCertificate) {
PersistentCertificate pc = (PersistentCertificate) obj;
equals = (this.commonName.equals(pc.commonName)
&& this.serial.equals(pc.serial) && this.b64Encoding
.equals(pc.b64Encoding));
}
return equals;
}

@Override
public int hashCode() {
return (this.b64Encoding + String.valueOf(this.serial) + this.commonName)
.hashCode();
}

protected Long getId() {
return id;
}

protected void setId(Long id) {
this.id = id;
}

public PersistentCertificationAuthority getIssuer() {
return issuer;
}

public void setIssuer(PersistentCertificationAuthority issuer) {
this.issuer = issuer;
}

public String getB64Encoding() {
return b64Encoding;
}

public void setB64Encoding(String encoding) {
this.b64Encoding = encoding;
}
```

```
public String getB64SubjKeyId() {
return b64SubjKeyId;
}

public void setB64SubjKeyId(String subjKeyId) {
this.b64SubjKeyId = subjKeyId;
}

public Date getNotAfter() {
return notAfter;
}

public void setNotAfter(Date notAfter) {
this.notAfter = notAfter;
}

}
package br.ufsc.jpki.persistence;

import java.io.IOException;
import java.math.BigInteger;
import java.security.cert.CertificateEncodingException;
import java.security.cert.CertificateException;
import java.security.cert.X509Extension;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;

import org.bouncycastle.asn1.ASN1Encodable;
import org.bouncycastle.asn1.DEREncodable;
import org.bouncycastle.asn1.x509.SubjectKeyIdentifier;
import org.bouncycastle.asn1.x509.X509Extensions;
import org.bouncycastle.x509.extension.SubjectKeyIdentifierStructure;
import org.bouncycastle.x509.extension.X509ExtensionUtil;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import sun.security.x509.SubjectKeyIdentifierExtension;
import br.ufsc.jpki.ca.CertificationAuthority;
import br.ufsc.jpki.exception.DataLoadingException;
import br.ufsc.jpki.exception.DataWritingException;
```

```
import br.ufsc.jpki.persistence.dao.CertificationAuthorityDAO;
import br.ufsc.jpki.util.Base64;

public class HCertificationAuthorityDAO implements CertificationAuthorityDAO {

// TODO criar metodo updateCA

public CertificationAuthority loadCA(String label)
throws DataLoadingException {
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
PersistentCertificationAuthority pca = (PersistentCertificationAuthority) session
.get(PersistentCertificationAuthority.class, label);
CertificationAuthority ca = null;
if (pca != null) {
try {
ca = pca.toCertificationAuthority();
} catch (Exception ex) {
throw new DataLoadingException(ex.getLocalizedMessage(), ex);
}
}
session.getTransaction().commit();
return ca;
}

public PersistentCertificationAuthority loadPCA(String label)
throws DataLoadingException {
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
PersistentCertificationAuthority pca = (PersistentCertificationAuthority) session
.load(PersistentCertificationAuthority.class, label);
session.getTransaction().commit();
return pca;
}

public Collection<String> loadCAsLabels() throws DataLoadingException {
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
Query query = session
.createQuery("select ca.label from PersistentCertificationAuthority as ca");
List<String> result = (List<String>) query.list();
session.getTransaction().commit();
return result;
}
```

```
public void storeCA(CertificationAuthority ca,
CertificationAuthority issuingCA) throws DataWritingException {
PersistentCertificationAuthority pca = null;
try {
pca = PersistentCertificationAuthority
.fromCertificationAuthority(ca);
if (issuingCA != null) {
pca.getCertificate().setIssuer(
PersistentCertificationAuthority
.fromCertificationAuthority(issuingCA));
} else {
pca.getCertificate().setIssuer(pca);
}
} catch (Exception e) {
throw new DataWritingException(e.getLocalizedMessage(), e);
}
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
Transaction t = session.beginTransaction();
session.persist(pca);
t.commit();
}

public CertificationAuthority loadCA(byte[] keyId, BigInteger serial)
throws DataLoadingException {
Iterator<PersistentCertificationAuthority> pcas = null;
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
Transaction t = session.beginTransaction();
String serialClause = "";
if (serial != null) {
serialClause = " and ca.certificate.serial = " + serial;
}

try {
Query q = session
.createQuery("from PersistentCertificationAuthority as ca"
+ " where ca.certificate.b64SubjKeyId = '"
+ Base64.encodeBytes(keyId) + "'" + serialClause);
pcas = (Iterator<PersistentCertificationAuthority>) q.iterate();
} catch (Exception e) {
throw new DataLoadingException(e.getLocalizedMessage(), e);
}
PersistentCertificationAuthority pca = null;
PersistentCertificationAuthority element = null;
```

```
CertificationAuthority ca = null;
if (pcas != null && pcas.hasNext()) {
pca = pcas.next();
while (pcas.hasNext()) {
element = pcas.next();
if (element.getCertificate().getNotAfter().compareTo(
pca.getCertificate().getNotAfter()) > 0) {
pca = element;
}
}
try {
ca = pca.toCertificationAuthority();
} catch (Exception ex) {
throw new DataLoadingException(ex.getLocalizedMessage(), ex);
}
}
t.commit();
return ca;
}

public boolean exists(String caLabel) throws DataLoadingException {
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
PersistentCertificationAuthority pca = (PersistentCertificationAuthority) session
.get(PersistentCertificationAuthority.class, caLabel);
session.getTransaction().commit();
return (pca != null);
}

public void updateCA(CertificationAuthority ca) throws DataWritingException {
try {
Session session = HibernateUtil.getSessionFactory()
.getCurrentSession();
session.beginTransaction();

byte[] binSubjKeyId = ca.getCertificate().getExtensionValue(
X509Extensions.SubjectKeyIdentifier.getId());
if (binSubjKeyId == null) {
throw new CertificateException(
"No subject key identifier found.");
}
SubjectKeyIdentifierStructure subjKeyId = null;
try {
subjKeyId = new SubjectKeyIdentifierStructure(binSubjKeyId);
```

```
} catch (IOException e) {
throw new CertificateException(
"Invalid subject key identifier: "
+ e.getLocalizedMessage(), e);
}

Query q = session
.createQuery(
"from PersistentCertificate as pc where pc.serial = ? and pc.b64SubjKeyId = ?")
.setBigInteger(0, ca.getCertificate().getSerialNumber())
.setString(1,
Base64.encodeBytes(subjKeyId.getKeyIdentifier()));

PersistentCertificate pc = (PersistentCertificate) q.uniqueResult();
if (pc == null) {
throw new DataWritingException(
"Unable to update certification authority due to missing certificate");
}

PersistentCertificationAuthority pca = (PersistentCertificationAuthority) session
.get(PersistentCertificationAuthority.class, ca.getLabel());
pca.setLastCertSerial(ca.getLastCertSerial());
pca.setLastCrlSerial(ca.getLastCrlSerial());
pca.setCertificate(pc);
session.getTransaction().commit();
} catch (Exception e) {
throw new DataWritingException("Error updating CA: "
+ e.getLocalizedMessage(), e);
}
}

public Collection<CertificationAuthority> loadCAs()
throws DataLoadingException {
Collection<CertificationAuthority> cas = new ArrayList<CertificationAuthority>();
Iterator<PersistentCertificationAuthority> iter = null;
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
Transaction t = session.beginTransaction();
try {
Query q = session
.createQuery("from PersistentCertificationAuthority");
iter = (Iterator<PersistentCertificationAuthority>) q.iterate();
while (iter.hasNext()) {
cas.add(iter.next().toCertificationAuthority());
}
```

```
} catch (Exception e) {
throw new DataLoadingException(e.getLocalizedMessage(), e);
}
t.commit();
return cas;
}

public Collection<CertificationAuthority> loadSubordinatedCAs(String caLabel)
throws DataLoadingException {
Collection<CertificationAuthority> casList = new LinkedList<CertificationAuthority>();
try {
Session session = HibernateUtil.getSessionFactory()
.getCurrentSession();
session.beginTransaction();
Query q = session
.createQuery(
"select pca from PersistentCertificationAuthority as pca, RevokedCertificate as rc where pca.certificate.issue
.setString(0, caLabel);
Iterator<PersistentCertificationAuthority> iter = q.iterate();
while (iter.hasNext()) {
PersistentCertificationAuthority pca = (PersistentCertificationAuthority) iter
.next();
casList.add(pca.toCertificationAuthority());
}
session.getTransaction().commit();
} catch (Exception e) {
throw new DataLoadingException(
"Error loading subordinated Certification authorities: "
+ e.getLocalizedMessage(), e);
}
return casList;
}

}
package br.ufsc.jpki.persistence;

import java.io.ByteArrayInputStream;
import java.math.BigInteger;
import java.security.cert.CertificateFactory;
import java.security.cert.X509CRL;
import java.security.cert.X509Certificate;

import org.hibernate.Query;
import org.hibernate.Session;
```

```
import br.ufsc.jpki.exception.DataLoadingException;
import br.ufsc.jpki.exception.DataWritingException;
import br.ufsc.jpki.persistence.dao.CertificateRevocationListDAO;
import br.ufsc.jpki.util.Base64;

public class HCertificateRevocationListDAO implements
CertificateRevocationListDAO {

public X509CRL loadCRL(BigInteger serial, String caLabel)
throws DataLoadingException {
X509CRL crl = null;
try {
Session session = HibernateUtil.getSessionFactory()
.getCurrentSession();
session.beginTransaction();
Query q = session
.createQuery("from PersistentCRL as pcrl where pcrl.serial = "
+ serial
+ " and pcrl.issuer.label = '"
+ caLabel
+ "'");
PersistentCRL pcrl = (PersistentCRL) q.uniqueResult();
if (pcrl != null) {
crl = pcrl.toX509CRL();
}
session.getTransaction().commit();
} catch (Exception e) {
throw new DataLoadingException("Error loading Certificate: "
+ e.getLocalizedMessage(), e);
}
return crl;
}

public void storeCRL(X509CRL crl, String caLabel)
throws DataWritingException {
PersistentCertificationAuthority pca = null;
try {
HCertificationAuthorityDAO caDao = new HCertificationAuthorityDAO();
pca = caDao.loadPCA(caLabel);
} catch (DataLoadingException e) {
throw new DataWritingException(
"Error loading the issuer certification auhtority: "
+ e.getLocalizedMessage(), e);
```

```
}
try {
PersistentCRL pcrl = PersistentCRL.fromX509CRL(crl);
pcrl.setIssuer(pca);
Session session = HibernateUtil.getSessionFactory()
.getCurrentSession();
session.beginTransaction();
session.save(pcrl);
session.getTransaction().commit();
} catch (Exception e) {
throw new DataWritingException("Error storing CRL: "
+ e.getLocalizedMessage(), e);
}
}

}
package br.ufsc.jpki.persistence;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.math.BigInteger;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.PrivateKey;
import java.security.cert.CertificateEncodingException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;

import org.bouncycastle.x509.extension.AuthorityKeyIdentifierStructure;

import br.ufsc.jpki.ca.CertificationAuthority;
import br.ufsc.jpki.ca.CertificationAuthority.AuthKeyIdType;
import br.ufsc.jpki.exception.DataLoadingException;
import br.ufsc.jpki.persistence.dao.CertificationAuthorityDAO;

public class PersistentCertificationAuthority {

private String label;

private int level;

private int authKeyIdType;
```

```
private BigInteger lastCertSerial;

private BigInteger lastCrlSerial;

private PersistentCertificate certificate;

private byte[] binaryKeyStore;

private static final KeyStore.PasswordProtection ksPasswd = new KeyStore.PasswordProtection(
"Hdj53264%%%$@%46KJrefhe*78".toCharArray());

public PersistentCertificationAuthority() {

}

public String getLabel() {
return label;
}

public void setLabel(String label) {
this.label = label;
}

public BigInteger getLastCertSerial() {
return lastCertSerial;
}

public void setLastCertSerial(BigInteger lastCertSerial) {
this.lastCertSerial = lastCertSerial;
}

public BigInteger getLastCrlSerial() {
return lastCrlSerial;
}

public void setLastCrlSerial(BigInteger lastCrlSerial) {
this.lastCrlSerial = lastCrlSerial;
}

public PersistentCertificate getCertificate() {
return certificate;
}

public void setCertificate(PersistentCertificate certificate) {
```

```
this.certificate = certificate;
}

public static PersistentCertificationAuthority fromCertificationAuthority(
CertificationAuthority ca) throws CertificateException,
KeyStoreException {
X509Certificate caCert = ca.getCertificate();
X509Certificate[] certs = { caCert };
PersistentCertificationAuthority pca = new PersistentCertificationAuthority();
ByteArrayOutputStream binOutStream = new ByteArrayOutputStream();
KeyStore ks = KeyStore.getInstance(KeyStore.getDefaultType());

try {
ks.load(null, ksPasswd.getPassword());
ks.setKeyEntry("prk", ca.getPrivateKey(), ksPasswd.getPassword(),
certs);
ks.store(binOutStream, ksPasswd.getPassword());
} catch (Exception e) {
throw new KeyStoreException(e.getLocalizedMessage(), e);
}

pca.setLabel(ca.getLabel());
pca.setLevel(ca.getLevel());
pca.setLastCertSerial(ca.getLastCertSerial());
pca.setLastCrlSerial(ca.getLastCrlSerial());
pca.setBinaryKeyStore(binOutStream.toByteArray());
PersistentCertificate pc = PersistentCertificate
.fromX509Certificate(caCert);
pca.setCertificate(pc);
pca.setAuthKeyIdType(ca.getAuthKeyIdType().toInt());
return pca;
}

public CertificationAuthority toCertificationAuthority()
throws CertificateEncodingException, CertificateException,
IOException {
ByteArrayInputStream binInStream = new ByteArrayInputStream(
this.binaryKeyStore);
PrivateKey prk = null;
try {
KeyStore ks = KeyStore.getInstance(KeyStore.getDefaultType());
ks.load(binInStream, ksPasswd.getPassword());
KeyStore.PrivateKeyEntry prkEntry = (KeyStore.PrivateKeyEntry) ks
.getEntry("prk", ksPasswd);
```

```
prk = prkEntry.getPrivateKey();
} catch (Exception ex) {
// TODO handle exception
ex.printStackTrace();
throw new RuntimeException(ex.getMessage());
}
CertificationAuthority.AuthKeyIdType aktype = CertificationAuthority.AuthKeyIdType
.fromInt(this.authKeyIdType);
CertificationAuthority ca = new CertificationAuthority(this.getLabel(),
this.level, this.getCertificate().toX509Certificate(), prk,
aktype);
ca.setLastCertSerial(this.getLastCertSerial());
ca.setLastCrlSerial(this.getLastCrlSerial());
ca.setCertificate(this.certificate.toX509Certificate());
return ca;
}

public byte[] getBinaryKeyStore() {
return binaryKeyStore;
}

public void setBinaryKeyStore(byte[] binaryKeyStore) {
this.binaryKeyStore = binaryKeyStore;
}

public boolean equals(Object obj) {
boolean equals = false;
if (obj instanceof PersistentCertificationAuthority) {
PersistentCertificationAuthority pca = (PersistentCertificationAuthority) obj;
equals = (this.label.equals(pca.label)
&& this.lastCertSerial.equals(pca.lastCertSerial)
&& this.lastCrlSerial.equals(pca.lastCrlSerial)
&& this.binaryKeyStore == pca.binaryKeyStore && this.certificate
.equals(pca.certificate));
}
return equals;
}

@Override
public int hashCode() {
return this.label.hashCode();
}

public int getAuthKeyIdType() {
```

```
return authKeyIdType;
}

public void setAuthKeyIdType(int authKeyIdType) {
this.authKeyIdType = authKeyIdType;
}

public int getLevel() {
return level;
}

public void setLevel(int level) {
this.level = level;
}

}
package br.ufsc.jpki.persistence;

import java.sql.SQLException;

import org.hibernate.*;
import org.hibernate.cfg.*;

public class HibernateUtil {
private static final SessionFactory sessionFactory;
static {
try {
// Create the SessionFactory from hibernate.cfg.xml
sessionFactory = new Configuration().configure()
.buildSessionFactory();
} catch (Throwable ex) {
// Make sure you log the exception, as it might be swallowed
System.err.println("Initial SessionFactory creation failed." + ex);
throw new ExceptionInInitializerError(ex);
}
}

public static SessionFactory getSessionFactory() {
return sessionFactory;
}

public static void cleanDatabase() {
final String SQL1 = "set referential_integrity false";
final String SQL2 = "delete from CertificationAuthorities";
```

```
final String SQL3 = "delete from Certificates";
final String SQL4 = "set referential_integrity true";

Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
session.createSQLQuery(SQL1).executeUpdate();
session.createSQLQuery(SQL2).executeUpdate();
session.createSQLQuery(SQL3).executeUpdate();
session.createSQLQuery(SQL4).executeUpdate();
session.getTransaction().commit();
}

public static void shutdown() throws SQLException {
sessionFactory.close();
}
}
package br.ufsc.jpki.persistence;

import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.math.BigInteger;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.cert.X509CRL;
import java.security.cert.X509Certificate;
import java.util.Date;

import org.bouncycastle.asn1.x509.X509Extensions;
import org.bouncycastle.x509.extension.SubjectKeyIdentifierStructure;

import br.ufsc.jpki.util.Base64;

public class RevokedCertificate {

private int id;

private PersistentCertificate certificate;

private int reasonCode;

private Date revocationDate;

public RevokedCertificate() {
}
```

```
public int getReasonCode() {
return reasonCode;
}

public void setReasonCode(int reasonCode) {
this.reasonCode = reasonCode;
}

public Date getRevocationDate() {
return revocationDate;
}

public void setRevocationDate(Date revocationDate) {
this.revocationDate = revocationDate;
}

@Override
public boolean equals(Object obj) {
boolean equals = false;
if (obj instanceof RevokedCertificate) {
RevokedCertificate rc = (RevokedCertificate) obj;
equals = this.certificate.equals(rc.certificate);
}
return equals;
}

@Override
public int hashCode() {
// TODO Auto-generated method stub
return this.certificate.hashCode();
}

// public static RevokedCertificate fromX509Certificate(X509Certificate
// x509Cert, int reasonCode, Date revocationDate) throws
// CertificateException {
// RevokedCertificate rc = new RevokedCertificate();
// rc.setSerial(x509Cert.getSerialNumber());
// rc.setCommonName(x509Cert.getSubjectDN().getName());
// rc.setNotAfter(x509Cert.getNotAfter());
// byte[] binSubjKeyId =
// x509Cert.getExtensionValue(X509Extensions.SubjectKeyIdentifier.getId());
// if (binSubjKeyId == null) {
// throw new CertificateException("No subject key identifier found.");
```

```
// }
// SubjectKeyIdentifierStructure subjKeyId = null;
// try {
// subjKeyId = new SubjectKeyIdentifierStructure(binSubjKeyId);
// }
// catch (IOException e) {
// throw new CertificateException("Invalid subject key identifier: " +
// e.getLocalizedMessage(), e);
// }
// rc.setB64SubjKeyId(Base64.encodeBytes(subjKeyId.getKeyIdentifier()));
// rc.setB64Encoding(Base64.encodeBytes(x509Cert.getEncoded()));
// rc.setReasonCode(reasonCode);
// rc.setRevocationDate(revocationDate);
// return rc;
// }

public int getId() {
return id;
}

public void setId(int id) {
this.id = id;
}

public PersistentCertificate getCertificate() {
return certificate;
}

public void setCertificate(PersistentCertificate certificate) {
this.certificate = certificate;
}

public BigInteger getSerial() {
BigInteger serial = null;
if (this.certificate != null)
serial = this.certificate.getSerial();
return serial;
}

}
package br.ufsc.jpki.persistence;

import java.io.ByteArrayInputStream;
import java.io.IOException;
```

```
import java.math.BigInteger;
import java.security.cert.CRLException;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.cert.X509CRL;
import java.util.Date;
import java.util.Set;

import org.bouncycastle.asn1.DERInteger;
import org.bouncycastle.asn1.DERNumericString;
import org.bouncycastle.asn1.x509.CRLNumber;
import org.bouncycastle.asn1.x509.X509Extensions;
import org.bouncycastle.x509.extension.X509ExtensionUtil;

import sun.security.x509.CRLNumberExtension;

import br.ufsc.jpki.util.Base64;

public class PersistentCRL {

private Long id;

private BigInteger serial;

private String b64Encoding;

private Date thisUpdate;

private Date nextUpdate;

private PersistentCertificationAuthority issuer;

public PersistentCRL() {

}

public String getB64Encoding() {
return b64Encoding;
}

public void setB64Encoding(String encoding) {
b64Encoding = encoding;
}
```

```
public Long getId() {
return id;
}

public void setId(Long id) {
this.id = id;
}

public PersistentCertificationAuthority getIssuer() {
return issuer;
}

public void setIssuer(PersistentCertificationAuthority issuer) {
this.issuer = issuer;
}

public Date getNextUpdate() {
return nextUpdate;
}

public void setNextUpdate(Date nextUpdate) {
this.nextUpdate = nextUpdate;
}

public Date getThisUpdate() {
return thisUpdate;
}

public void setThisUpdate(Date thisUpdate) {
this.thisUpdate = thisUpdate;
}

public boolean equals(Object obj) {
boolean equals = false;
if (obj instanceof PersistentCRL) {
PersistentCRL pcrl = (PersistentCRL) obj;
equals = (this.serial.equals(pcrl.serial)
&& this.issuer.equals(pcrl.issuer)
&& this.b64Encoding.equals(pcrl.b64Encoding)
&& this.thisUpdate.equals(pcrl.thisUpdate) && this.nextUpdate
.equals(pcrl.nextUpdate));
}
return equals;
}
```

```
public BigInteger getSerial() {
return serial;
}

public void setSerial(BigInteger serial) {
this.serial = serial;
}

public X509CRL toX509CRL() throws CRLException, CertificateException,
IOException {
ByteArrayInputStream inStream = new ByteArrayInputStream(Base64
.decode(this.getB64Encoding()));
CertificateFactory certFactory = CertificateFactory.getInstance("X509");
X509CRL crl = (X509CRL) certFactory.generateCRL(inStream);
inStream.close();
return crl;
}

public static PersistentCRL fromX509CRL(X509CRL crl) throws CRLException,
IOException {
PersistentCRL pcrl = new PersistentCRL();
pcrl.setB64Encoding(Base64.encodeBytes(crl.getEncoded()));
pcrl.setNextUpdate(crl.getNextUpdate());
pcrl.setThisUpdate(crl.getThisUpdate());
Set<String> oids = crl.getNonCriticalExtensionOIDs();
if (oids.contains(X509Extensions.CRLNumber.getId())) {
DERInteger crlSerial = (DERInteger) X509ExtensionUtil
.fromExtensionValue(crl
.getExtensionValue(X509Extensions.CRLNumber.getId()));
pcrl.setSerial(crlSerial.getPositiveValue());
}
return pcrl;
}

}
package br.ufsc.jpki.persistence;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import br.ufsc.jpki.exception.ConnectionFailedException;

public class DatabaseManager {
```

```
private static Connection connection = null;

static {
try {
Class.forName("org.hsqldb.jdbcDriver");
} catch (java.lang.ClassNotFoundException ex) {
System.err.println("Error loading JDBC Driver: " + ex.getMessage());
}
}

public static Connection getConnection() throws ConnectionFailedException {
if (DatabaseManager.connection == null) {
try {
DatabaseManager.connection = DriverManager
.getConnection("jdbc:hsqldb:hsql:///home/jeanms/workspace/data/database");
} catch (SQLException ex) {
throw new ConnectionFailedException(
"Error retrieving connection to database: "
+ ex.getMessage());
}
}
return DatabaseManager.connection;
}
}
package br.ufsc.jpki.container;

import java.util.Calendar;
import java.util.Vector;

public class CertParameters {

private String internalSubject;

private Calendar notBefore;

private Calendar notAfter;

private String policyOID;

private String cpsURI;

private String crlDP;
```

```
private String issuingCALabel;

private Vector<KeyPurpose> keyUsages;

private String authKeyIdType;

private String pathLen;

public enum KeyPurpose {
cRLSign, dataEncipherment, decipherOnly, digitalSignature, encipherOnly, keyAgreement, keyCertSign, keyEnciphe
}

public CertParameters() {
this.internalSubject = "|";
}

public CertParameters(String l, String s, Calendar notBefore,
Calendar notAfter) {
this.internalSubject = s;
this.notBefore = notBefore;
this.notAfter = notAfter;
}

public String getSubject() {
String subject = this.internalSubject.substring(1,
(this.internalSubject.length() - 1));
return subject.replace("|", ", ");
}

public Calendar getNotBefore() {
return this.notBefore;
}

public void setNotBefore(Calendar notBefore) {
this.notBefore = notBefore;
}

public Calendar getNotAfter() {
return notAfter;
}

public void setNotAfter(Calendar notAfter) {
this.notAfter = notAfter;
}
```

```
public void setCommonName(String cn) {
if (cn != null && !"".equals(cn)) {
if (!this.internalSubject.contains("CN=")) {
this.internalSubject += "CN=" + cn + "|";
} else {
this.internalSubject = this.internalSubject.replaceFirst(
"CN=[\\s*\\w*\\s*]*", "CN=" + cn);
}
}
}

public void setOrganization(String o) {
if (o != null && !"".equals(o)) {
if (!this.internalSubject.contains("O=")) {
this.internalSubject += "O=" + o + "|";
} else {
this.internalSubject = this.internalSubject.replaceFirst(
"O=[\\s*\\w*\\s*]*", "O=" + o);
}
}
}

public void setOrganizationUnit(String ou) {
if (ou != null && !"".equals(ou)) {
if (!this.internalSubject.contains("OU=")) {
this.internalSubject += "OU=" + ou + "|";
} else {
this.internalSubject = this.internalSubject.replaceFirst(
"OU=[\\s*\\w*\\s*]*", "OU=" + ou);
}
}
}

public void setCountry(String c) {
if (c != null && !"".equals(c)) {
if (!this.internalSubject.contains("C=")) {
this.internalSubject += "C=" + c + "|";
} else {
this.internalSubject = this.internalSubject.replaceFirst(
"C=[\\s*\\w*\\s*]*", "C=" + c);
}
}
}
```

```
public void setStateProvincy(String st) {
if (st != null && !"".equals(st)) {
if (!this.internalSubject.contains("ST=")) {
this.internalSubject += "ST=" + st + "|";
} else {
this.internalSubject = this.internalSubject.replaceFirst(
"ST=[\\s*\\w*\\s*]*", "ST=" + st);
}
}
}

public void setLocality(String l) {
if (l != null && !"".equals(l)) {
if (!this.internalSubject.contains("L=")) {
this.internalSubject += "L=" + l + "|";
} else {
this.internalSubject = this.internalSubject.replaceFirst(
"L=[\\s*\\w*\\s*]*", "L=" + l);
}
}
}

public String getCpsURI() {
return cpsURI;
}

public void setCpsURI(String cpsURI) {
if (cpsURI != null && !"".equals(cpsURI))
this.cpsURI = cpsURI;
}

public String getCrlDP() {
return crlDP;
}

public void setCrlDP(String crlDP) {
if (crlDP != null && !"".equals(crlDP))
this.crlDP = crlDP;
}

public String getPolicyOID() {
return policyOID;
}
```

```
public void setPolicyOID(String policyOID) {
if (policyOID != null && !"".equals(policyOID))
this.policyOID = policyOID;
}

public void addKeyUsage(KeyPurpose k) {
if (this.keyUsages == null) {
this.keyUsages = new Vector<KeyPurpose>();
}
if (!this.keyUsages.contains(k)) {
this.keyUsages.add(k);
}
}

public Vector<KeyPurpose> getKeyUsages() {
return this.keyUsages;
}

public void setIssuingCALabel(String l) {
this.issuingCALabel = l;
}

public String getIssuingCALabel() {
return this.issuingCALabel;
}

public String getAuthKeyIdType() {
return authKeyIdType;
}

public void setAuthKeyIdType(String authKeyIdType) {
this.authKeyIdType = authKeyIdType;
}

public String getPathLen() {
return pathLen;
}

public void setPathLen(String pathLen) {
this.pathLen = pathLen;
}

}
```

```
package br.ufsc.jpki.container;

public class CAParameters {

private String label;

private CertParameters certParams;

public CertParameters getCertParams() {
return certParams;
}

public void setCertParams(CertParameters certParams) {
if (!"".equals(this.certParams))
this.certParams = certParams;
}

public String getLabel() {
return label;
}

public void setLabel(String label) {
if (!"".equals(this.certParams))
this.label = label;
}

}
package br.ufsc.jpki;

import java.io.IOException;
import java.math.BigInteger;
import java.security.InvalidKeyException;
import java.security.KeyPair;
import java.security.KeyStore;
import java.security.NoSuchAlgorithmException;
import java.security.SignatureException;
import java.security.cert.CRLException;
import java.security.cert.CertificateException;
import java.security.cert.X509CRL;
import java.security.cert.X509Certificate;
import java.util.Calendar;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;
```

```
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.UnsupportedCallbackException;

import org.bouncycastle.asn1.x509.AlgorithmIdentifier;
import org.bouncycastle.asn1.x509.AuthorityKeyIdentifier;
import org.bouncycastle.asn1.x509.SubjectPublicKeyInfo;
import org.bouncycastle.asn1.x509.X509Extensions;
import org.bouncycastle.x509.extension.AuthorityKeyIdentifierStructure;
import org.bouncycastle.x509.extension.SubjectKeyIdentifierStructure;

import br.ufsc.jpki.ca.CertificationAuthority;
import br.ufsc.jpki.ca.CertificationAuthorityBuilder;
import br.ufsc.jpki.container.CAParameters;
import br.ufsc.jpki.container.CertParameters;
import br.ufsc.jpki.container.CertParameters.KeyPurpose;
import br.ufsc.jpki.exception.CANotFoundException;
import br.ufsc.jpki.exception.CertificationAuthorityBuildingException;
import br.ufsc.jpki.exception.CertificationAuthorityException;
import br.ufsc.jpki.exception.DataLoadingException;
import br.ufsc.jpki.exception.DataWritingException;
import br.ufsc.jpki.io.FileHandler;
import br.ufsc.jpki.io.ConfigurationFileReader;
import br.ufsc.jpki.io.IOHandler;
import br.ufsc.jpki.persistence.HCertificateDAO;
import br.ufsc.jpki.persistence.HCertificateRevocationListDAO;
import br.ufsc.jpki.persistence.HCertificationAuthorityDAO;
import br.ufsc.jpki.persistence.RevokedCertificate;
import br.ufsc.jpki.persistence.dao.CertificateDAO;
import br.ufsc.jpki.persistence.dao.CertificateRevocationListDAO;
import br.ufsc.jpki.persistence.dao.CertificationAuthorityDAO;
import br.ufsc.jpki.provider.CryptographicServicesProvider;
import br.ufsc.jpki.provider.CryptographicServicesProvider.KeyType;
import br.ufsc.jpki.util.Base64;

public class PKIManager {

private static PKIManager pkim = null;

private final String defaultPwd = "1234";

CryptographicServicesProvider provider;

private PKIManager() {
```

```
this.provider = new CryptographicServicesProvider();
}

public static PKIManager getInstance() throws InstantiationException {
if (PKIManager.pkim == null) {
PKIManager.pkim = new PKIManager();
}
return pkim;
}

public CertificationAuthority createCA(CAParameters params)
throws CertificationAuthorityBuildingException {

CertificationAuthorityDAO caDao = new HCertificationAuthorityDAO();
boolean exist = false;
try {
exist = caDao.exists(params.getLabel());
} catch (DataLoadingException e) {
throw new CertificationAuthorityBuildingException(e
.getLocalizedMessage(), e);
}
if (exist) {
throw new CertificationAuthorityBuildingException(
"There is already a CA with this label");
}
CertificationAuthorityBuilder caBuilder = new CertificationAuthorityBuilder(
this.provider);
CertificationAuthority ica = null;
try {
caBuilder.setLabel(params.getLabel());
caBuilder.setCertParameters(params.getCertParams());
if (!params.getCertParams().getIssuingCALabel().equals(
params.getLabel())) {
ica = caDao.loadCA(params.getCertParams().getIssuingCALabel());
if (ica == null) {
throw new CertificationAuthorityBuildingException(
"Issuer ca does not exist");
}
}
caBuilder.setIssuingCA(ica);
} catch (DataLoadingException e) {
throw new CertificationAuthorityBuildingException(e
.getLocalizedMessage(), e);
}
```

```
CertificationAuthority ca = caBuilder.build();
try {
if (ica != null) {
caDao.updateCA(ica);
}
caDao.storeCA(ca, ica);
} catch (DataWritingException e) {
throw new CertificationAuthorityBuildingException(e
.getLocalizedMessage(), e);
}
return ca;
}

public void revokeCertificate(String serial, String caLabel,
String reasonCode) throws CertificateException {
try {
CertificationAuthorityDAO caDao = new HCertificationAuthorityDAO();
CertificationAuthority ca = caDao.loadCA(caLabel);
ca.revokeCertificate(new BigInteger(serial), Integer
.parseInt(reasonCode));
} catch (DataLoadingException e) {
throw new CertificateException("Issuer CA not found: "
+ e.getLocalizedMessage(), e);
}
}

public X509CRL generateCRL(String caLabel, int validity)
throws DataLoadingException, CANotFoundException, CRLException {

CertificationAuthorityDAO cadao = new HCertificationAuthorityDAO();
CertificationAuthority currentCA = cadao.loadCA(caLabel);
if (currentCA == null) {
throw new CANotFoundException("CA " + caLabel + " not found");
}
CertificateDAO cdao = new HCertificateDAO();
List<RevokedCertificate> rcs = cdao.loadRevokedCertificates(Calendar
.getInstance().getTime(), caLabel);
X509CRL crl = null;
try {
crl = currentCA.issueCRL(rcs, validity);
CertificateRevocationListDAO crldao = new HCertificateRevocationListDAO();
crldao.storeCRL(crl, currentCA.getLabel());
cadao.updateCA(currentCA);
} catch (Exception e) {
```

```
throw new CRLException(
"Error in CRL generation: " + e.getMessage(), e);
}
return crl;
}

public KeyStore createEndEntity(CertParameters cp)
throws DataLoadingException, CANotFoundException,
CertificateException, DataWritingException,
NoSuchAlgorithmException {

CertificationAuthorityDAO cadao = new HCertificationAuthorityDAO();
CertificationAuthority currentCA = cadao.loadCA(cp.getIssuingCALabel());
if (currentCA == null) {
throw new CANotFoundException("CA " + cp.getIssuingCALabel()
+ " not found");
}

KeyPair kp = this.provider.generateKeyPair(KeyType.RSA);
X509Certificate cert = null;
try {
cert = currentCA.issueCertificate(cp, kp.getPublic());
} catch (Exception e) {
throw new CertificateException(
"Error issuing end entity certificate: "
+ e.getLocalizedMessage(), e);
}

CertificateDAO cdao = new HCertificateDAO();
cdao.storeCertificate(cert, currentCA.getLabel());

KeyStore ks = null;
try {
String pwd = IOHandler
.readFromStdin("Enter a password to protect PKCS#12 content (4 to 8 digits)");
if (pwd == null || pwd.length() < 4 || pwd.length() > 8) {
throw new CertificateException(
"The password length must be between 4 and 8 digits");
}
ks = KeyStore.getInstance("PKCS12");
ks.load(null, pwd.toCharArray());
X509Certificate[] certs = { cert };
ks.setKeyEntry("entity", kp.getPrivate(), pwd.toCharArray(), certs);
} catch (Exception e) {
```

```
throw new CertificateException(
"Error generating end entity PKCS#12: "
+ e.getLocalizedMessage(), e);
}

cadao.updateCA(currentCA);
return ks;
}

public X509Certificate renewCertificate(X509Certificate oldCert)
throws CertificateException, InvalidKeyException,
SignatureException, CANotFoundException, DataLoadingException,
DataWritingException {

byte[] encodedKeyId = null;
BigInteger serial = null;

if (oldCert.getSubjectDN().equals(oldCert.getIssuerDN())) { // self
// signed
encodedKeyId = oldCert
.getExtensionValue(X509Extensions.SubjectKeyIdentifier
.getId());
if (encodedKeyId == null) {
throw new CANotFoundException(
"Unable to load subject key id for a self signed certificate,\n missing Subject Key Identifier extension.");
}
SubjectKeyIdentifierStructure subjKeyId = null;
try {
subjKeyId = new SubjectKeyIdentifierStructure(encodedKeyId);
encodedKeyId = subjKeyId.getKeyIdentifier();
} catch (IOException e) {
throw new CANotFoundException(
"Unable to load subject key id for a self signed certificate,\n"
+ " error decoding Subject Key Identifier extension: "
+ e.getLocalizedMessage(), e);
}
} else {
encodedKeyId = oldCert
.getExtensionValue(X509Extensions.AuthorityKeyIdentifier
.getId());
if (encodedKeyId == null) {
throw new CANotFoundException(
"Unable to load issuer CA, missing Authority Key Identifier extension.");
}
```

```
try {
AuthorityKeyIdentifierStructure authKeyId = new AuthorityKeyIdentifierStructure(
encodedKeyId);
encodedKeyId = authKeyId.getKeyIdentifier();
serial = authKeyId.getAuthorityCertSerialNumber();
} catch (IOException e) {
throw new CANotFoundException(
"Unable to load issuer CA, error decoding Authority Key Identifier extension: "
+ e.getLocalizedMessage(), e);
}
}

CertificationAuthorityDAO cadao = new HCertificationAuthorityDAO();
CertificationAuthority currentCA = cadao.loadCA(encodedKeyId, serial);

if (currentCA == null) {
throw new CANotFoundException(
"Unable to load issuer CA matching the auhtority key identifier.");
}

try {
oldCert.checkValidity(Calendar.getInstance().getTime());
oldCert.verify(currentCA.getCertificate().getPublicKey());
} catch (Exception e) {
throw new CertificateException(
"Validity check or verification failed for input certificate: "
+ e.getLocalizedMessage(), e);
}

X509Certificate newCert = null;
try {
Calendar notAfter = Calendar.getInstance();
if (currentCA.getCertificate().getSubjectDN().equals(
currentCA.getCertificate().getIssuerDN())) {
notAfter.add(Calendar.YEAR, 10);
} else {
int certValidityPeriod = oldCert.getNotAfter().getYear()
- oldCert.getNotBefore().getYear();
int caValidityPeriod = currentCA.getCertificate().getNotAfter()
.getYear()
- currentCA.getCertificate().getNotBefore().getYear();
int caValidityLeft = currentCA.getCertificate().getNotAfter()
.getYear()
- notAfter.getTime().getYear();
```

```
if (caValidityPeriod < certValidityPeriod) {
notAfter.add(Calendar.YEAR, caValidityLeft);
} else {
notAfter.add(Calendar.YEAR, certValidityPeriod);
}
}
newCert = currentCA.renewCertificate(oldCert, notAfter);
} catch (Exception e) {
throw new CertificateException("Error in certificate renewal: "
+ e.getMessage(), e);
}

CertificateDAO cdao = new HCertificateDAO();
cdao.storeCertificate(newCert, currentCA.getLabel());
if (currentCA.getCertificate().equals(oldCert)) {
currentCA.setCertificate(newCert);
}
if (oldCert.getBasicConstraints() != -1) { // not a CA certificate
currentCA.revokeCertificate(oldCert.getSerialNumber(), 4); // reason
// code
// =
// superseded
}
cadao.updateCA(currentCA);
return newCert;
}

private void issueEndEntityCertificates(CertificationAuthority ca,
Collection<String> subjects, String outputPath)
throws CertificateException, DataWritingException, IOException,
NoSuchAlgorithmException {

Iterator<String> iter = subjects.iterator();
System.out.println("Issuing " + subjects.size() + " certificates to "
+ ca.getLabel());
CertParameters cp = new CertParameters();
cp.setCountry("BR");
cp.setStateProvincy("SC");
cp.setLocality("Florianopolis");
cp.setIssuingCALabel(ca.getLabel());
cp.setOrganization("TEST");
cp.setOrganizationUnit(ca.getLabel());
cp.setNotBefore(Calendar.getInstance());
Calendar notAfter = Calendar.getInstance();
```

```
cp.addKeyUsage(KeyPurpose.digitalSignature);
cp.addKeyUsage(KeyPurpose.nonRepudiation);
cp.addKeyUsage(KeyPurpose.dataEncipherment);
notAfter.add(Calendar.YEAR, 1);
cp.setNotAfter(notAfter);
CertificateDAO cdao = new HCertificateDAO();
CertificationAuthorityDAO cadao = new HCertificationAuthorityDAO();
int i = 0;
while (iter.hasNext()) {
System.out.println("Issuing: " + (i + 1) + "/" + subjects.size()
+ " to " + ca.getLabel());
cp.setCommonName(iter.next());
KeyPair kp = this.provider.generateKeyPair(KeyType.RSA);
X509Certificate cert = null;
try {
cert = ca.issueCertificate(cp, kp.getPublic());
} catch (Exception e) {
throw new CertificateException(
"Error issuing end entity certificate: "
+ e.getLocalizedMessage(), e);
}

cdao.storeCertificate(cert, ca.getLabel());

KeyStore ks = null;
try {
ks = KeyStore.getInstance("PKCS12");
ks.load(null, defaultPwd.toCharArray());
X509Certificate[] certs = { cert };
ks.setKeyEntry("entity_" + i, kp.getPrivate(), defaultPwd
.toCharArray(), certs);
} catch (Exception e) {
throw new CertificateException(
"Error generating end entity PKCS#12: "
+ e.getLocalizedMessage(), e);
}

cadao.updateCA(ca);
FileHandler.storeKeyStore(outputPath + "/entity_" + i++ + ".p12",
ks, defaultPwd.toCharArray());
}
}

public void copyCA(String oldCaLabel, String newCaLabel,
```

```
String targetCALabel, String output)
throws CertificationAuthorityException {
CertificationAuthorityDAO cadao = new HCertificationAuthorityDAO();
CertificationAuthority subCA = null;
CertificationAuthority targetCA = null;
try {
subCA = cadao.loadCA(oldCaLabel);
targetCA = cadao.loadCA(targetCALabel);
} catch (Exception e) {
throw new CertificationAuthorityException("CA" + oldCaLabel
+ "not found: " + e.getLocalizedMessage(), e);
}

CertificationAuthority newCA = targetCA.addSubordinate(newCaLabel,
subCA);

try {
cadao.storeCA(newCA, targetCA);
cadao.updateCA(targetCA);
FileHandler.storeCertificate(output, newCA.getCertificate());
} catch (Exception e) {
throw new CertificationAuthorityException(
"Failed to store new CA: " + e.getLocalizedMessage(), e);
}
}

// public void replaceCA(String oldCALabel, String newCALabel, String
// outputDir) throws CertificationAuthorityBuildingException {
// try {
// CertificationAuthorityDAO cadao = new HCertificationAuthorityDAO();
// CertificationAuthority oldCA = cadao.loadCA(oldCALabel);
// CertificationAuthorityBuilder builder = new
// CertificationAuthorityBuilder();
// CertificationAuthority newCA = null; //builder.buildFromCA(newCALabel,
// oldCA);
// cadao.storeCA(newCA, null);
// Collection<CertificationAuthority> subCAs =
// cadao.loadSubordinatedCAs(oldCALabel);
// Iterator<CertificationAuthority> iter = subCAs.iterator();
// CertificationAuthority newSubCA = null;
// while (iter.hasNext()) {
// CertificationAuthority ca = (CertificationAuthority) iter.next();
// builder = new CertificationAuthorityBuilder();
// newSubCA = builder.buildFromCA(ca.getLabel() + "-new");
```

```
// X509Certificate newCert = newCA.issueCertificate(ca.getCertificate());
// ca.setCertificate(newCert);
// }
// }
// catch (Exception e) {
// throw new
// CertificationAuthorityBuildingException(e.getLocalizedMessage(), e);
// }
// }

public void executeTask(String[] task) throws RuntimeException {
boolean taskOk = false;
String errorMessage = "Invalid command";
if ("newca".equals(task[0])) {
if (task.length == 3) {
taskOk = true;
try {
CAParameters caparams = ConfigurationFileReader
.loadCAParameters(task[1]);
CertificationAuthority ca = this.createCA(caparams);
FileHandler.storeCertificate(task[2], ca.getCertificate());
} catch (Exception e) {
throw new RuntimeException(e.getLocalizedMessage(), e);
}
} else {
errorMessage = "Incorrect usage\n\t\tUsage: newca <caconfig> <output>";
}
} else if ("newee".equals(task[0])) {
if (task.length == 3) {
taskOk = true;
try {
CertParameters certparams = ConfigurationFileReader
.loadCertParameters(task[1]);
KeyStore ks = this.createEndEntity(certparams);
FileHandler.storeKeyStore(task[2], ks, this.defaultPwd
.toCharArray());
} catch (Exception e) {
throw new RuntimeException(e.getLocalizedMessage(), e);
}
} else {
errorMessage = "Incorrect usage\n\t\tUsage: newee <certconfig> <output>";
}
} else if ("renew".equals(task[0])) {
if (task.length == 2) {
```

```
taskOk = true;
try {
X509Certificate newCert = this.renewCertificate(FileHandler
.loadCertificate(task[1]));
FileHandler.storeCertificate(task[1], newCert);
} catch (Exception e) {
throw new RuntimeException(e.getLocalizedMessage(), e);
}
} else {
errorMessage = "Incorrect usage\n\t\tUsage: renew <certfile>";
}
} else if ("revoke".equals(task[0])) {
if (task.length == 4) {
taskOk = true;
try {
this.revokeCertificate(task[1], task[2], task[3]);
} catch (Exception e) {
throw new RuntimeException(e.getLocalizedMessage(), e);
}
} else {
errorMessage = "Incorrect usage\n\t\tUsage: revoke <certserial> <calabel> <reasoncode>";
}
} else if ("newcrl".equals(task[0])) {
if (task.length == 4) {
taskOk = true;
try {
X509CRL crl = this.generateCRL(task[1], Integer
.parseInt(task[2]));
FileHandler.storeCRL(task[3], crl);
} catch (NumberFormatException e) {
throw new RuntimeException("Parameter " + task[2]
+ " must be an integer.", e);
} catch (Exception e) {
throw new RuntimeException(e.getLocalizedMessage(), e);
}
} else {
errorMessage = "Incorrect usage\n\t\tUsage: newcrl <calabel> <validity> <outputpath>";
}
} else if ("simulate".equals(task[0])) {
CertificationAuthorityDAO cadao = new HCertificationAuthorityDAO();
if (task.length == 4) {
taskOk = true;
int first = 0, numCerts = 0;
try {
```

```
numCerts = Integer.parseInt(task[2]);
} catch (NumberFormatException e) {
throw new RuntimeException(e.getLocalizedMessage(), e);
}

if (!FileHandler.isValidDir(task[3])) {
throw new RuntimeException("Output Directory is not valid.");
}

try {
Collection<CertificationAuthority> cas = cadao.loadCAs();
Iterator<CertificationAuthority> iter = cas.iterator();
String outputPath = "";
while (iter.hasNext()) {
CertificationAuthority ca = iter.next();
if (ca.getLevel() == 2
&& ca.getLastCertSerial().compareTo(
new BigInteger("1000")) < 0) {
Collection<String> names = FileHandler.loadNames(
task[1], first, numCerts);
outputPath = FileHandler.makeDir(task[3], ca
.getLabel());
if (names != null) {
this.issueEndEntityCertificates(ca, names,
outputPath);
first += numCerts;
} else {
throw new RuntimeException(
"Invalid file or too few names.");
}
}
}
} catch (Exception e) {
throw new RuntimeException(e.getLocalizedMessage(), e);
}
} else {
errorMessage = "Incorrect usage\n\t\tUsage: simulate <namesfile> <numcerts> <outputpath>";
}
} else if ("copyca".equals(task[0])) {
if (task.length == 5) {
taskOk = true;
// if (!FileHandler.isValidDir(task[4]))
// {
// throw new RuntimeException("Output Directory is not valid.");
```

```
// }
try {
this.copyCA(task[1], task[2], task[3], task[4]);
} catch (Exception e) {
throw new RuntimeException(e.getLocalizedMessage(), e);
}
} else {
errorMessage = "Incorrect usage\n\t\tUsage: copyca <ca-name> <new-ca-name> <target-ca> <output>";
}
} else if ("help".equals(task[0])) {
taskOk = true;
this.printHelp();
}
if (!taskOk) {
throw new RuntimeException(errorMessage);
}
}

public void printHelp() {
System.out.println("Usage: jpki [command]");
System.out.println(" commands: ");
System.out
.println(" \tnewca\t\t<caconfig>\t<output>\t\t\tcreate a new ca based on the configuration file.");
System.out
.println(" \tnewee\t\t<certconfig>\t<output>\t\t\tcreate a new end entity based on the configuration file.");
System.out
.println(" \trenew\t\t<certfile>\t\t\t\t\trenew a issued certificate and store into certifile.");
System.out
.println(" \trevoke\t\t<certserial>\t<calabel>\t<reasoncode>\trenew a issued certificate and store into certif
System.out
.println(" \tnewcrl\t\t<calabel>\t<validity>\t<outputpath>\tissue a new CRL for the specified CA with validity
System.out
.println(" \tsimulate\t<namesfile>\t<numcerts>\t<outputpath>\tsimulate a complete PKI issuing numcerts certifi
System.out.println(" \thelp\t\t\t\t\t\t\tprint this help and exit.");
}

public void handle(Callback[] callbacks) throws IOException,
UnsupportedCallbackException {
// for (int i = 0; i < callbacks.length; i++) {
// if (callbacks[i] instanceof PasswordCallback) {
// PasswordCallback pwdCallback = (PasswordCallback) callbacks[i];
// String pin = this.ui.getPIN();
// if (pin != null) {
// pwdCallback.setPassword(pin.toCharArray());
```

```
// }
// else {
// throw new IOException("User cancelled authentication");
// }
// }
// else {
// throw new UnsupportedCallbackException(callbacks[i], "Unrecognized
// Callback");
// }
// }
}

}
package br.ufsc.jpki.io;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.StringReader;
import java.nio.charset.Charset;
import java.security.KeyStore;
import java.security.cert.CRLException;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.cert.X509CRL;
import java.security.cert.X509Certificate;
import java.util.ArrayList;
import java.util.Collection;
import java.util.StringTokenizer;

import org.bouncycastle.asn1.ASN1InputStream;
import org.bouncycastle.asn1.ASN1Sequence;
import org.bouncycastle.asn1.DERObject;
import org.bouncycastle.asn1.DERSequence;
import org.bouncycastle.asn1.pkcs.CertificationRequest;
import org.bouncycastle.asn1.pkcs.CertificationRequestInfo;
import org.hsqldb.lib.StringInputStream;
```

```
public class FileHandler {

public static X509Certificate loadCertificate(String certFilePath)
throws FileNotFoundException, IOException, CertificateException,
IllegalArgumentException {
FileInputStream inputStream = FileHandler
.openFileForReading(certFilePath);
X509Certificate certificate = null;
CertificateFactory certFactory = CertificateFactory.getInstance("X509");
certificate = ((X509Certificate) certFactory
.generateCertificate(inputStream));
inputStream.close();
return certificate;
}

public static X509Certificate loadCertificate(InputStream inputStream)
throws CertificateException, IOException {
X509Certificate certificate = null;
CertificateFactory certFactory = CertificateFactory.getInstance("X509");
certificate = ((X509Certificate) certFactory
.generateCertificate(inputStream));
inputStream.close();
return certificate;
}

public static void storeCertificate(String filePath, Certificate cert)
throws CertificateException, IOException {
FileOutputStream outputStream = FileHandler
.openFileForWriting(filePath);
outputStream.write(cert.getEncoded());
outputStream.close();
}

public static CertificationRequest loadCertificateRequest(
String requestFilePath) throws FileNotFoundException, IOException {
FileInputStream inputStream = FileHandler
.openFileForReading(requestFilePath);
ASN1InputStream asn1InputStream = new ASN1InputStream(inputStream);
ASN1Sequence asn1Sequence = (DERSequence) asn1InputStream.readObject();
CertificationRequest certReq = new CertificationRequest(asn1Sequence);
asn1InputStream.close();
inputStream.close();
return certReq;
```

```
}

public static CertificationRequest loadCertificateRequest(
InputStream inputStream) throws CertificateException, IOException {
ASN1InputStream asn1InputStream = new ASN1InputStream(inputStream);
ASN1Sequence asn1Sequence = (DERSequence) asn1InputStream.readObject();
CertificationRequest certReq = new CertificationRequest(asn1Sequence);
asn1InputStream.close();
inputStream.close();
return certReq;
}

private static FileInputStream openFileForReading(String filePath)
throws FileNotFoundException, IOException {
File inputFile = new File(filePath);
if (!inputFile.exists()) {
throw new FileNotFoundException("File \"" + filePath
+ "\" not found");
}
if (!inputFile.canRead()) {
throw new IOException("Permission denied reading file \""
+ filePath + "\"");
}
return new FileInputStream(inputFile);
}

private static FileOutputStream openFileForWriting(String filePath)
throws IOException {
File outputFile = new File(filePath);
if (outputFile.exists()) {
if (!outputFile.delete()) {
throw new IOException("Permission denied writing file \""
+ filePath + "\"");
}
}
if (!outputFile.createNewFile()) {
throw new IOException("Permission denied writing file \""
+ filePath + "\"");
}
return new FileOutputStream(outputFile);
}

public static void storeKeyStore(String filePath, KeyStore ks, char[] pwd)
throws IOException {
```

```
FileOutputStream outputStream = FileHandler
.openFileForWriting(filePath);
try {
ks.store(outputStream, pwd);
} catch (Exception e) {
throw new IOException(e.getLocalizedMessage());
}
outputStream.close();
}

public static Collection<String> loadNames(String filePath, int startLine,
int numNames) throws FileNotFoundException, IOException {
InputStream input = FileHandler.openFileForReading(filePath);
Collection<String> names = new ArrayList<String>();
BufferedReader reader = new BufferedReader(new InputStreamReader(input,
"UTF8"));
for (int i = 0; i < startLine; i++) {
reader.readLine();
}
String line = null;
boolean eof = false;
while (names.size() < numNames && !eof) {
line = reader.readLine();
if (line != null) {
if (line.length() != 0) {
names.add(line.trim());
}
} else {
eof = true;
}
}
return names;
}

public static boolean isValidDir(String path) {
File outputFile = new File(path);
return (outputFile.isDirectory() && outputFile.canWrite());
}

public static String makeDir(String path, String dirName)
throws IOException {
File outputFile = new File(path);
File newDir = null;
if (outputFile.isDirectory() && outputFile.canWrite()) {
```

```
newDir = new File(outputFile + "/" + dirName);
if (!newDir.exists()) {
if (!newDir.mkdir())
throw new IOException("Error in directory creation.");
}
}
return newDir.getAbsolutePath();
}

public static void storeCRL(String filePath, X509CRL crl)
throws CRLException, IOException {
FileOutputStream outputStream = FileHandler
.openFileForWriting(filePath);
outputStream.write(crl.getEncoded());
outputStream.close();
}

}
package br.ufsc.jpki.io;

import java.io.File;
import java.io.IOException;
import java.util.Calendar;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

import br.ufsc.jpki.container.CAParameters;
import br.ufsc.jpki.container.CertParameters;
import br.ufsc.jpki.container.CertParameters.KeyPurpose;
import br.ufsc.jpki.exception.InvalidConfigFileException;

public class ConfigurationFileReader {

private static Document config;
```

```
private static void loadDocument(String fileName)
throws InvalidConfigFileException, IOException {
File configFile = new File(fileName);
if (configFile.exists() && configFile.canRead()) {
DocumentBuilder b = null;
try {
b = DocumentBuilderFactory.newInstance().newDocumentBuilder();
} catch (ParserConfigurationException e) {
throw new IOException(e.getLocalizedMessage());
}

try {
config = b.parse(configFile);
} catch (SAXException e) {
throw new InvalidConfigFileException(e.getLocalizedMessage(), e);
}

} else {
throw new IOException("File not found or not readable in path: "
+ fileName);
}
}

public static CAParameters loadCAParameters(String fileName)
throws InvalidConfigFileException, IOException {
ConfigurationFileReader.loadDocument(fileName);
CAParameters caparams = new CAParameters();
NodeList nodes = config.getChildNodes();
Node caNode = null;
for (int i = 0; i < nodes.getLength(); i++) {
if ("ca".equals(nodes.item(i).getNodeName())) {
caNode = nodes.item(i);
break;
}
}

if (caNode == null) {
throw new InvalidConfigFileException(
"No ca node found in config file");
}
NamedNodeMap caAttributes = caNode.getAttributes();
if (caAttributes == null && caAttributes.getNamedItem("name") == null) {
throw new InvalidConfigFileException(
"No ca name attribute found in config file");
```

```
}
Node nameAttribute = caAttributes.getNamedItem("name");
if (nameAttribute.getNodeValue() != null) {
caparams.setLabel(nameAttribute.getNodeValue());
} else {
throw new InvalidConfigFileException(
"No valid ca name found in config file");
}

NodeList caChildNodes = caNode.getChildNodes();
Node certNode = null;
for (int i = 0; i < caChildNodes.getLength(); i++) {
if ("certificate".equals(caChildNodes.item(i).getNodeName())) {
certNode = caChildNodes.item(i);
break;
}
}
if (certNode == null) {
throw new InvalidConfigFileException(
"No certificate node found in config file");
}
CertParameters certparams = ConfigurationFileReader
.loadCertParameters(certNode);
caparams.setCertParams(certparams);
return caparams;
}

public static CertParameters loadCertParameters(String fileName)
throws InvalidConfigFileException, IOException {
ConfigurationFileReader.loadDocument(fileName);
NodeList nodes = config.getChildNodes();
Node certNode = null;
for (int i = 0; i < nodes.getLength(); i++) {
if ("certificate".equals(nodes.item(i).getNodeName())) {
certNode = nodes.item(i);
break;
}
}
if (certNode == null) {
throw new InvalidConfigFileException(
"No certificate node found in config file");
}
return ConfigurationFileReader.loadCertParameters(certNode);
}
```

```
private static CertParameters loadCertParameters(Node certNode)
throws InvalidConfigFileException {
CertParameters cp = new CertParameters();
NodeList certParams = certNode.getChildNodes();
for (int i = 0; i < certParams.getLength(); i++) {
if ("issuer".equals(certParams.item(i).getNodeName())) {
Element issuerNode = (Element) certParams.item(i);
if (issuerNode.getAttribute("name") == null) {
throw new InvalidConfigFileException(
"No issuer name attribute found in config file");
}
cp.setIssuingCALabel(issuerNode.getAttribute("name"));
} else if ("subject".equals(certParams.item(i).getNodeName())) {
NodeList subjChildNodes = certParams.item(i).getChildNodes();
for (int j = 0; j < subjChildNodes.getLength(); j++) {
if ("cn".equals(subjChildNodes.item(j).getNodeName())) {
cp.setCommonName(((Element) subjChildNodes.item(j))
.getTextContent());
} else if ("o".equals(subjChildNodes.item(j).getNodeName())) {
cp.setOrganization(((Element) subjChildNodes.item(j))
.getTextContent());
} else if ("ou"
.equals(subjChildNodes.item(j).getNodeName())) {
cp.setOrganizationUnit(((Element) subjChildNodes
.item(j)).getTextContent());
} else if ("c".equals(subjChildNodes.item(j).getNodeName())) {
cp.setCountry(((Element) subjChildNodes.item(j))
.getTextContent());
} else if ("st"
.equals(subjChildNodes.item(j).getNodeName())) {
cp.setStateProvincy(((Element) subjChildNodes.item(j))
.getTextContent());
} else if ("l".equals(subjChildNodes.item(j).getNodeName())) {
cp.setLocality(((Element) subjChildNodes.item(j))
.getTextContent());
}
}
} else if ("validity".equals(certParams.item(i).getNodeName())) {
Element validity = (Element) certParams.item(i);
if (validity.getTextContent() != null) {
Calendar notBefore = Calendar.getInstance();
Calendar notAfter = (Calendar) notBefore.clone();
cp.setNotBefore(notBefore);
```

```
try {
notAfter.add(Calendar.YEAR, Integer.parseInt(validity
.getTextContent()));
} catch (NumberFormatException e) {
throw new InvalidConfigFileException(
"No valid validity found in config file", e);
}
cp.setNotAfter(notAfter);
}
} else if ("extensions".equals(certParams.item(i).getNodeName())) {
NodeList extensionList = certParams.item(i).getChildNodes();
for (int j = 0; j < extensionList.getLength(); j++) {
Element currentExtension = null;
if ("extension".equals(extensionList.item(j).getNodeName())) {
currentExtension = (Element) extensionList.item(j);
if (currentExtension.getAttribute("type") == null) {
throw new InvalidConfigFileException(
"Invalid extension type");
}
if ("keyUsage".equals(currentExtension
.getAttribute("type"))) {
NodeList usages = currentExtension.getChildNodes();
for (int k = 0; k < usages.getLength(); k++) {
if ("digitalSignature".equals(usages.item(k)
.getNodeName())) {
cp.addKeyUsage(KeyPurpose.digitalSignature);
} else if ("nonRepudiation".equals(usages.item(
k).getNodeName())) {
cp.addKeyUsage(KeyPurpose.nonRepudiation);
} else if ("keyEncipherment".equals(usages
.item(k).getNodeName())) {
cp.addKeyUsage(KeyPurpose.keyEncipherment);
} else if ("dataEncipherment".equals(usages
.item(k).getNodeName())) {
cp.addKeyUsage(KeyPurpose.dataEncipherment);
} else if ("keyCertSign".equals(usages.item(k)
.getNodeName())) {
cp.addKeyUsage(KeyPurpose.keyCertSign);
} else if ("cRLSign".equals(usages.item(k)
.getNodeName())) {
cp.addKeyUsage(KeyPurpose.cRLSign);
} else if ("encipherOnly".equals(usages.item(k)
.getNodeName())) {
cp.addKeyUsage(KeyPurpose.encipherOnly);
```

```
} else if ("decipherOnly".equals(usages.item(k)
.getNodeName())) {
cp.addKeyUsage(KeyPurpose.decipherOnly);
} else if ("keyAgreement".equals(usages.item(k)
.getNodeName())) {
cp.addKeyUsage(KeyPurpose.keyAgreement);
}
}
} else if ("certificatePolicy".equals(currentExtension
.getAttribute("type"))) {
NodeList policyInfos = currentExtension
.getChildNodes();
for (int k = 0; k < policyInfos.getLength(); k++) {
if ("oid".equals(policyInfos.item(k)
.getNodeName())) {
cp.setPolicyOID(((Element) policyInfos
.item(k)).getTextContent());
} else if ("cps".equals(policyInfos.item(k)
.getNodeName())) {
cp
.setCpsURI(((Element) policyInfos
.item(k)).getTextContent());
}
}
} else if ("crlDistributionPoint"
.equals(currentExtension.getAttribute("type"))) {
NodeList crlDPList = currentExtension
.getChildNodes();
for (int k = 0; k < crlDPList.getLength(); k++) {
if ("uri".equals(crlDPList.item(k)
.getNodeName())) {
cp.setCrlDP(((Element) crlDPList.item(k))
.getTextContent());
}
}
} else if ("authKeyId".equals(currentExtension
.getAttribute("type"))) {
NodeList authKeyIdList = currentExtension
.getChildNodes();
for (int k = 0; k < authKeyIdList.getLength(); k++) {
if ("type".equals(authKeyIdList.item(k)
.getNodeName())) {
cp
.setAuthKeyIdType(((Element) authKeyIdList
```

```
.item(k)).getTextContent());
}
}
} else if ("caPathLen".equals(currentExtension
.getAttribute("type"))) {
cp.setPathLen(currentExtension.getTextContent());
}
}
}
}
}
return cp;
}

}
package br.ufsc.jpki.io;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class IOHandler {
public static String readFromStdin(String message) throws IOException {
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
String input = null;
System.out.print(message + ": ");
input = in.readLine();
return input;
}
}
package br.ufsc.jpki.exception;

public class DataLoadingException extends Exception {

public DataLoadingException() {
super();
// TODO Auto-generated constructor stub
}

public DataLoadingException(String message, Throwable cause) {
super(message, cause);
// TODO Auto-generated constructor stub
}
```

```
public DataLoadingException(String message) {
super(message);
// TODO Auto-generated constructor stub
}

public DataLoadingException(Throwable cause) {
super(cause);
// TODO Auto-generated constructor stub
}

}
package br.ufsc.jpki.exception;

public class DataWritingException extends Exception {

public DataWritingException() {
super();
// TODO Auto-generated constructor stub
}

public DataWritingException(String message, Throwable cause) {
super(message, cause);
// TODO Auto-generated constructor stub
}

public DataWritingException(String message) {
super(message);
// TODO Auto-generated constructor stub
}

public DataWritingException(Throwable cause) {
super(cause);
// TODO Auto-generated constructor stub
}

}
package br.ufsc.jpki.exception;

public class CertificateNotFoundException extends Exception {

}
package br.ufsc.jpki.exception;

public class PKCS11Exception extends Exception {
```

```
public PKCS11Exception() {
super();
// TODO Auto-generated constructor stub
}

public PKCS11Exception(String arg0, Throwable arg1) {
super(arg0, arg1);
// TODO Auto-generated constructor stub
}

public PKCS11Exception(String arg0) {
super(arg0);
// TODO Auto-generated constructor stub
}

public PKCS11Exception(Throwable arg0) {
super(arg0);
// TODO Auto-generated constructor stub
}

}
package br.ufsc.jpki.exception;

public class CANotFoundException extends Exception {

public CANotFoundException() {
// TODO Auto-generated constructor stub
}

public CANotFoundException(String arg0) {
super(arg0);
// TODO Auto-generated constructor stub
}

public CANotFoundException(Throwable arg0) {
super(arg0);
// TODO Auto-generated constructor stub
}

public CANotFoundException(String arg0, Throwable arg1) {
super(arg0, arg1);
// TODO Auto-generated constructor stub
}
```

```
}
package br.ufsc.jpki.exception;

public class CertificationAuthorityException extends Exception {

public CertificationAuthorityException() {
// TODO Auto-generated constructor stub
}

public CertificationAuthorityException(String arg0) {
super(arg0);
// TODO Auto-generated constructor stub
}

public CertificationAuthorityException(Throwable arg0) {
super(arg0);
// TODO Auto-generated constructor stub
}

public CertificationAuthorityException(String arg0, Throwable arg1) {
super(arg0, arg1);
// TODO Auto-generated constructor stub
}

}
package br.ufsc.jpki.exception;

public class ConnectionFailedException extends Exception {
public ConnectionFailedException() {
super();
}
public ConnectionFailedException(String msg) {
super(msg);
}
public ConnectionFailedException(String message, Throwable cause) {
super(message, cause);
}
public ConnectionFailedException(Throwable cause) {
super(cause);
}
}
package br.ufsc.jpki.exception;
```

```
public class InvalidConfigFileException extends Exception {

public InvalidConfigFileException() {
super();
// TODO Auto-generated constructor stub
}
public InvalidConfigFileException(String message, Throwable cause) {
super(message, cause);
// TODO Auto-generated constructor stub
}
public InvalidConfigFileException(String message) {
super(message);
// TODO Auto-generated constructor stub
}
public InvalidConfigFileException(Throwable cause) {
super(cause);
// TODO Auto-generated constructor stub
}

}
package br.ufsc.jpki.exception;

public class CertificationAuthorityBuildingException extends Exception {

public CertificationAuthorityBuildingException() {
super();
}
public CertificationAuthorityBuildingException(String msg) {
super(msg);
}
public CertificationAuthorityBuildingException(String message,
Throwable cause) {
super(message, cause);
}
public CertificationAuthorityBuildingException(Throwable cause) {
super(cause);
}
}
```

Apêndice B

Código Fonte Renovador de Certificados

```
<?php
include ’IOException.php’;

class UploadHandler
{
    /**
 * $_FILES[’userfile’][’name’]
 * The original name of the file on the client machine.
 * $_FILES[’userfile’][’type’]
 * The mime type of the file, if the browser provided this information. An example would be "image/gif". This r
 * $_FILES[’userfile’][’size’]
 * The size, in bytes, of the uploaded file.
 * $_FILES[’userfile’][’tmp_name’]
 * The temporary filename of the file in which the uploaded file was stored on the server.
 * $_FILES[’userfile’][’error’]
 *
 * UPLOAD_ERR_OK
 * Value: 0; There is no error, the file uploaded with success.
 * UPLOAD_ERR_INI_SIZE
 * Value: 1; The uploaded file exceeds the upload_max_filesize directive in php.ini.
 * UPLOAD_ERR_FORM_SIZE
 * Value: 2; The uploaded file exceeds the MAX_FILE_SIZE directive that was specified in the HTML form.
 * UPLOAD_ERR_PARTIAL
 * Value: 3; The uploaded file was only partially uploaded.
 * UPLOAD_ERR_NO_FILE
```

```
 * Value: 4; No file was uploaded.
 * UPLOAD_ERR_NO_TMP_DIR
 * Value: 6; Missing a temporary folder. Introduced in PHP 4.3.10 and PHP 5.0.3.
 * UPLOAD_ERR_CANT_WRITE
 * Value: 7; Failed to write file to disk. Introduced in PHP 5.1.0.
 **/

protected $file;
protected $path;

protected function __construct($file = array()) {
$this->file = $file;
$this->allowed_types = array();
$this->allowed_extensions = array();
$this->path = $this->file['tmp_name'];
}
/**
 * @throw IOException
 **/
public static function getFile($name,$allowed = array()) {
if(!$_FILES[$name]['error']) {
$upFile = new UploadHandler($_FILES[$name]);
if(count($allowed) > 0 && !in_array($upFile->file['type'],$allowed)) {
throw new IOException("Erro tipo invalido: " . $upFile['type']);
}
return $upFile;
} else {
throw new IOException("Erro no upload: " . $_FILES[$name]['error']);
}
}

/**
 * @throw IOException
 **/
public function moveTo($path, $fileName="") {
// se filename eh vazio, coloca o nome do arquivo do upload
// faz verificacao de tipos e se necessario dispara exception
( $fileName!="" ? $name = $fileName : $name = $this->file['name'] );
if(!rename($this->path, $path . "/" . $name)) {
throw new IOException("Erro ao mover arquivo");
}
$this->path = $path . "/" . $name;
}
```

```
/**
 * @throw IOException
 **/
public function copyTo($path, $filename) {
( $fileName!="" ? $name = $fileName : $name = $this->file['name'] );
if(!copy($this->path,$path . "/" . $name)) {
throw new IOException("Erro ao copiar arquivo");
}
$this->path = $path . "/" . $name;
}

/**
 * @throw IOException
 **/
public function getContent() {
$content = file_get_contents($this->path);
if (!$content) {
throw IOException("Erro ao ler arquivo.");
}
return $content;
}

/**
 * @throw IOException
 **/
public function putContent($content) {
if (!file_put_contents($this->file['name'],$content) && $content != "") {
throw new IOExcepton("Erro ao escrever dados no arquivo.");
}
}

public function getName() {
return $this->file["name"];
}

public function getsize() {
return sizeof($this->path);
}

public function setValidTypes($types = array()) {
// dispara exception no move to se o tipo do arquivo for invalida
$this->allowed_types = $types;
}
```

```
public function getPath() {
return $this->path;
}

}
?>K 14
svn:executable
V 1
*
END
<?php
include 'IOException.php';

class UploadHandler
{
    /**
 * $_FILES['userfile']['name']
 * The original name of the file on the client machine.
 * $_FILES['userfile']['type']
 * The mime type of the file, if the browser provided this information. An example would be "image/gif". This r
 * $_FILES['userfile']['size']
 * The size, in bytes, of the uploaded file.
 * $_FILES['userfile']['tmp_name']
 * The temporary filename of the file in which the uploaded file was stored on the server.
 * $_FILES['userfile']['error']
 *
 * UPLOAD_ERR_OK
 * Value: 0; There is no error, the file uploaded with success.
 * UPLOAD_ERR_INI_SIZE
 * Value: 1; The uploaded file exceeds the upload_max_filesize directive in php.ini.
 * UPLOAD_ERR_FORM_SIZE
 * Value: 2; The uploaded file exceeds the MAX_FILE_SIZE directive that was specified in the HTML form.
 * UPLOAD_ERR_PARTIAL
 * Value: 3; The uploaded file was only partially uploaded.
 * UPLOAD_ERR_NO_FILE
 * Value: 4; No file was uploaded.
 * UPLOAD_ERR_NO_TMP_DIR
 * Value: 6; Missing a temporary folder. Introduced in PHP 4.3.10 and PHP 5.0.3.
 * UPLOAD_ERR_CANT_WRITE
 * Value: 7; Failed to write file to disk. Introduced in PHP 5.1.0.
 **/

protected $file;
protected $path;
```

```
protected function __construct($file = array()) {
$this->file = $file;
$this->allowed_types = array();
$this->allowed_extensions = array();
$this->path = $this->file['tmp_name'];
}

/**
 * @throw IOException
 **/
public static function getFile($name,$allowed = array()) {
if(!$_FILES[$name]['error']) {
$upFile = new UploadHandler($_FILES[$name]);
if(count($allowed) > 0 && !in_array($upFile->file['type'],$allowed)) {
throw new IOException("Erro tipo invalido: " . $upFile['type']);
}
return $upFile;
} else {
throw new IOException("Erro no upload: " . $_FILES[$name]['error']);
}
}

/**
 * @throw IOException
 **/
public function moveTo($path, $fileName="") {
// se filename eh vazio, coloca o nome do arquivo do upload
// faz verificacao de tipos e se necessario dispara exception
( $fileName!="" ? $name = $fileName : $name = $this->file['name'] );
if(!rename($this->path, $path . "/" . $name)) {
throw new IOException("Erro ao mover arquivo");
}
$this->path = $path . "/" . $name;
}

/**
 * @throw IOException
 **/
public function copyTo($path, $filename) {
( $fileName!="" ? $name = $fileName : $name = $this->file['name'] );
if(!copy($this->path,$path . "/" . $name)) {
throw new IOException("Erro ao copiar arquivo");
}
```

```
$this->path = $path . "/" . $name;
}

/**
 * @throw IOException
 **/
public function getContent() {
$content = file_get_contents($this->path);
if (!$content) {
throw IOException("Erro ao ler arquivo.");
}
return $content;
}

/**
 * @throw IOException
 **/
public function putContent($content) {
if (!file_put_contents($this->file[’name’],$content) && $content != "") {
throw new IOExcepton("Erro ao escrever dados no arquivo.");
}
}

public function getName() {
return $this->file["name"];
}

public function getsize() {
return sizeof($this->path);
}

public function setValidTypes($types = array()) {
// dispara exception no move to se o tipo do arquivo for invalida
$this->allowed_types = $types;
}

public function getPath() {
return $this->path;
}

}
?><?php
class IOException extends Exception {
```

```
}
?><?php
class CertificateRenewalException extends Exception {

}
?><?php
class IOException extends Exception {

}
?>K 14
svn:executable
V 1
*
END
<?php
class CertificateRenewalException extends Exception {

}
?><?php

class DownloadController extends Zend_Controller_Action {

private $session;

public function init()
{
$this->initView();
$this->view->baseUrl = $this->_request->getBaseUrl();
$this->session = new Zend_Session_Namespace();
}

public function indexAction() {
$this->render();
}

public function getCertAction() {
$this->view->filename = basename($this->session->certPath);
$this->view->filesize = filesize($this->session->certPath);
$this->view->certificate = file_get_contents($this->session->certPath);
$this->render();
}

}
?><?php
```

```
class IndexController extends Zend_Controller_Action {

public function init()
{
$this->initView();
$this->view->baseUrl = $this->_request->getBaseUrl();
}

public function indexAction()
{
$this->view->title = "Renove seu certificado";
$this->render();
}

public function errorAction() {
$this->view->errorMessage = $this->_request->getParam("cause");
$this->render();
}

}

?><?php
include 'Upload.php';
include 'CertificateRenewalException.php';

class CertificateController extends Zend_Controller_Action {

// if(strpos($_SERVER['HTTP_USER_AGENT'], 'MSIE') !== false)
// {
// header("Content-type: application/pkix-cert");
// }
// else
// {
// header("Content-type: application/x-x509-user-cert");
// }

private $session;

public function init()
{
$this->initView();
$this->view->baseUrl = $this->_request->getBaseUrl();
$this->session = new Zend_Session_Namespace();
}
```

```
public function indexAction() {
$this->render();
}

/**
 * @throw IOException
 **/
public function renewSSLAction() {
$filePath = sys_get_temp_dir() . "/endentity.crt";
$nb = file_put_contents($filePath, $_SERVER['REDIRECT_SSL_CLIENT_CERT']);
if ($nb == 0) {
$this->_forward("error", "index", null ,array("cause" => "Error " . $status . ": ". $message));
}
try {
$this->renewCertificate($filePath);
$this->session->certPath = $filePath;
$this->_forward("index","download",null);
}
catch(Exception $e) {
$this->_forward("error", "index", null ,array("cause" => $e->getMessage()));
}
}

public function showFormAction() {
$this->render();
}

public function submitFormAction() {
if ($this->_request->isPost()) {
try {
$upload = UploadHandler::getFile("upFile");
$upload->setValidTypes(array("application/x-x509-user-cert"));
$upload->moveTo(sys_get_temp_dir());
$this->sendChallengeMail($this->_request->getParam("mail"), $upload->getPath());
$this->session->certPath = $upload->getPath();
$this->render();
// $this->renewCertificate($upload->getPath());
// $this->_forward("index","download",null);
}
catch(Exception $e) {
$this->_forward("error", "index", null ,array("cause" => $e->getMessage()));
}
}
```

```
}

public function renewCertAction() {
if ($this->_request->getParam("challenge") == $this->session->challenge) {
$this->renewCertificate($this->session->certPath);
$this->_forward("index","download",null);
}
else {
$this->_forward("error", "index", null ,array("cause" => "Resposta errada ao desafio."));
}
}

private function sendChallengeMail($mailAddress, $certFile) {
$type = exec('file -bi ' . $certFile );
$certContent = file_get_contents($certFile);
if ($type == "application/octet-stream") { // if DER format, convert to PEM
$certContent = $this->der2pem($certContent);
file_put_contents($certFile,$certContent);
}

$randNumber = rand(1000000,1000000000000);
$this->session->challenge = $randNumber;

$certParse = openssl_x509_parse($certContent);
$subject = $certParse['subject'];
$issuer = $certParse['issuer'];
$mailContent =
"Este Ã© um email confidencial!\n" .
"Ele foi enviado a vocÃª como forma de confirmaÃ§Ã$o da posse da chave privada correspondente ao certificado:\n
"NÃºmero Serial: " . $certParse['serialNumber'] .
"Emitido para: " . $subject['C']  . ", "
 . $subject['ST'] . ", "
 . $subject['L']  . ", "
 . $subject['OU'] . ", "
 . $subject['O']  . ", "
 . $subject['CN'] . ", " .
"Emitido por: "  . $issuer['C']  . ", "
 . $issuer['OU'] . ", "
 . $issuer['O']  . ", "
 . $issuer['CN'] . ", " .
"Se vocÃª estÃ¡ visualizando esta mensagem Ã© porque possui a chave privada, logo basta recortar " .
"o nÃºmero abaixo e colocÃ¡-lo no formulÃ¡rio de renovaÃ§Ã$o" .
"--------------- recore aqui --------------\n" .
$randNumber .
```

```
"\n--------------- recore aqui --------------";

$plainMesgFile = sys_get_temp_dir() . "/msg.txt";
$encMesgFile = sys_get_temp_dir() . "/encmsg.txt";
file_put_contents($plainMesgFile,$mailContent);
if (openssl_pkcs7_encrypt($plainMesgFile, $encMesgFile, $certContent,
array("To" => $mailAddress,
  "From" => "nobody@certrenewer.com",
  "Subject" => "Prova de Posse"))
) {
$mailContent = file_get_contents($encMesgFile);
$parts = explode("\n\n", $mailContent, 2);
$this->sendMail("",$mailAddress, "Renovador", "nobody@certrenewer.com",
"Prova de Posse", $parts[1],$parts[0]);
}
unlink($plainMesgFile);
unlink($encMesgFile);
}

function sendMail($ToName, $ToEmail, $FromName, $FromEmail, $Subject, $Body, $Header)
{
$smtp = fsockopen("150.162.66.6", 25);

$InputBuffer = fgets($smtp, 1024);

fputs($smtp, "HELO sitename.com\n");
$InputBuffer = fgets($smtp, 1024);
fputs($smtp, "mail From: $FromEmail\n");
$InputBuffer = fgets($smtp, 1024);
fputs($smtp, "RCPT To: $ToEmail\n");
$InputBuffer = fgets($smtp, 1024);
fputs($smtp, "DATA\n");
$InputBuffer = fgets($smtp, 1024);
fputs($smtp, "$Header");
fputs($smtp, "From: $FromName <$FromEmail>\n");
fputs($smtp, "To: $ToName <$ToEmail>\n");
fputs($smtp, "Subject: $Subject\n\n");
fputs($smtp, "$Body\r\n.\r\n");
fputs($smtp, "QUIT\n");
$InputBuffer = fgets($smtp, 1024);
fclose($smtp);
}

private function renewCertificate($filePath) {
```

```
$message = system("/usr/local/jpki/jpki renew " . $filePath, $status);
if ($status) {
throw new CertificateRenewalException(
"Erro na renovaÃ§Ã$o do certificado - Erro " . $status . ": ". $message);
}
}

private function pem2der($pem_data) {
$begin = "CERTIFICATE-----";
$end   = "-----END";
$pem_data = substr($pem_data, strpos($pem_data, $begin)+strlen($begin));
$pem_data = substr($pem_data, 0, strpos($pem_data, $end));
$der = base64_decode($pem_data);
return $der;
}

private function der2pem($der_data) {
$pem = chunk_split(base64_encode($der_data), 64, "\n");
$pem = "-----BEGIN CERTIFICATE-----\n".$pem."-----END CERTIFICATE-----\n";
return $pem;
}

}
?><?php

class DownloadController extends Zend_Controller_Action {

private $session;

public function init()
{
$this->initView();
$this->view->baseUrl = $this->_request->getBaseUrl();
$this->session = new Zend_Session_Namespace();
}

public function indexAction() {
$this->render();
}

public function getCertAction() {
$this->view->filename = basename($this->session->certPath);
$this->view->filesize = filesize($this->session->certPath);
$this->view->certificate = file_get_contents($this->session->certPath);
```

```
$this->render();
}

}
?>

<?php
class IndexController extends Zend_Controller_Action {

public function init()
{
$this->initView();
$this->view->baseUrl = $this->_request->getBaseUrl();
}

public function indexAction()
{
$this->view->title = "Renove seu certificado";
$this->render();
}

public function errorAction() {
$this->view->errorMessage = $this->_request->getParam("cause");
$this->render();
}

}
?>

<?php
include 'Upload.php';
include 'CertificateRenewalException.php';

class CertificateController extends Zend_Controller_Action {

// if(strpos($_SERVER['HTTP_USER_AGENT'], 'MSIE') !== false)
// {
// header("Content-type: application/pkix-cert");
// }
// else
// {
// header("Content-type: application/x-x509-user-cert");
// }
```

```
private $session;

public function init()
{
$this->initView();
$this->view->baseUrl = $this->_request->getBaseUrl();
$this->session = new Zend_Session_Namespace();
}

public function indexAction() {
$this->render();
}

/**
 * @throw IOException
 **/
public function renewSSLAction() {
$filePath = sys_get_temp_dir() . "/endentity.crt";
$nb = file_put_contents($filePath, $_SERVER['REDIRECT_SSL_CLIENT_CERT']);
if ($nb == 0) {
$this->_forward("error", "index", null ,array("cause" => "Error " . $status . ": ". $message));
}
try {
$this->renewCertificate($filePath);
$this->session->certPath = $filePath;
$this->_forward("index","download",null);
}
catch(Exception $e) {
$this->_forward("error", "index", null ,array("cause" => $e->getMessage()));
}
}

public function showFormAction() {
$this->render();
}

public function submitFormAction() {
if ($this->_request->isPost()) {
try {
$upload = UploadHandler::getFile("upFile");
$upload->setValidTypes(array("application/x-x509-user-cert"));
$upload->moveTo(sys_get_temp_dir());
$this->sendChallengeMail($this->_request->getParam("mail"), $upload->getPath());
$this->session->certPath = $upload->getPath();
```

```
$this->render();
// $this->renewCertificate($upload->getPath());
// $this->_forward("index","download",null);
}
catch(Exception $e) {
$this->_forward("error", "index", null ,array("cause" => $e->getMessage()));
}
}
}

public function renewCertAction() {
if ($this->_request->getParam("challenge") == $this->session->challenge) {
$this->renewCertificate($this->session->certPath);
$this->_forward("index","download",null);
}
else {
$this->_forward("error", "index", null ,array("cause" => "Resposta errada ao desafio."));
}
}

private function sendChallengeMail($mailAddress, $certFile) {
$type = exec('file -bi ' . $certFile );
$certContent = file_get_contents($certFile);
if ($type == "application/octet-stream") { // if DER format, convert to PEM
$certContent = $this->der2pem($certContent);
file_put_contents($certFile,$certContent);
}

$randNumber = rand(1000000,1000000000000);
$this->session->challenge = $randNumber;

$certParse = openssl_x509_parse($certContent);
$subject = $certParse['subject'];
$issuer = $certParse['issuer'];
$mailContent =
"Este Ã© um email confidencial!\n" .
"Ele foi enviado a vocÃª como forma de confirmaÃ§Ã$o da posse da chave privada correspondente ao certificado:\n
"NÃºmero Serial: " . $certParse['serialNumber'] .
"Emitido para: " . $subject['C']  . ", "
 . $subject['ST'] . ", "
 . $subject['L']  . ", "
 . $subject['OU'] . ", "
 . $subject['O']  . ", "
 . $subject['CN'] . ", " .
```

```
"Emitido por: "  . $issuer[’C’]  . ", "
 . $issuer[’OU’] . ", "
 . $issuer[’O’]  . ", "
 . $issuer[’CN’] . ", " .
"Se vocÃª estÃ¡ visualizando esta mensagem Ã© porque possui a chave privada, logo basta recortar " .
"o nÃºmero abaixo e colocÃ¡-lo no formulÃ¡rio de renovaÃ§Ã$o" .
"--------------- recore aqui --------------\n" .
$randNumber .
"\n--------------- recore aqui --------------";

$plainMesgFile = sys_get_temp_dir() . "/msg.txt";
$encMesgFile = sys_get_temp_dir() . "/encmsg.txt";
file_put_contents($plainMesgFile,$mailContent);
if (openssl_pkcs7_encrypt($plainMesgFile, $encMesgFile, $certContent,
array("To" => $mailAddress,
  "From" => "nobody@certrenewer.com",
  "Subject" => "Prova de Posse"))
) {
$mailContent = file_get_contents($encMesgFile);
$parts = explode("\n\n", $mailContent, 2);
$this->sendMail("",$mailAddress, "Renovador", "nobody@certrenewer.com",
"Prova de Posse", $parts[1],$parts[0]);
}
unlink($plainMesgFile);
unlink($encMesgFile);
}

function sendMail($ToName, $ToEmail, $FromName, $FromEmail, $Subject, $Body, $Header)
{
$smtp = fsockopen("150.162.66.6", 25);

$InputBuffer = fgets($smtp, 1024);

fputs($smtp, "HELO sitename.com\n");
$InputBuffer = fgets($smtp, 1024);
fputs($smtp, "mail From: $FromEmail\n");
$InputBuffer = fgets($smtp, 1024);
fputs($smtp, "RCPT To: $ToEmail\n");
$InputBuffer = fgets($smtp, 1024);
fputs($smtp, "DATA\n");
$InputBuffer = fgets($smtp, 1024);
fputs($smtp, "$Header");
fputs($smtp, "From: $FromName <$FromEmail>\n");
fputs($smtp, "To: $ToName <$ToEmail>\n");
```

```
fputs($smtp, "Subject: $Subject\n\n");
fputs($smtp, "$Body\r\n.\r\n");
fputs($smtp, "QUIT\n");
$InputBuffer = fgets($smtp, 1024);
fclose($smtp);
}

private function renewCertificate($filePath) {
$message = system("/usr/local/jpki/jpki renew " . $filePath, $status);
if ($status) {
throw new CertificateRenewalException(
"Erro na renovaÃ§Ã$o do certificado - Erro " . $status . ": ". $message);
}
}

private function pem2der($pem_data) {
$begin = "CERTIFICATE-----";
$end   = "-----END";
$pem_data = substr($pem_data, strpos($pem_data, $begin)+strlen($begin));
$pem_data = substr($pem_data, 0, strpos($pem_data, $end));
$der = base64_decode($pem_data);
return $der;
}

private function der2pem($der_data) {
$pem = chunk_split(base64_encode($der_data), 64, "\n");
$pem = "-----BEGIN CERTIFICATE-----\n".$pem."-----END CERTIFICATE-----\n";
return $pem;
}

}
?>

<?php
header("Pragma: public");
# configura o tempo de expiracao da pagina
header("Expires: 0");
# faz com que o browser busque o arquivo no servidor e nao no cache
// header("Cache-Control: must-revalidate, post-check=0, pre-check=0");
header("Content-type: application/x-x509-user-cert");
// header("Content-Type: application/force-download");
// header("Content-Type: application/download");
// header("Content-Disposition: attachment; filename=" . $this->filename . ";");
header("Content-Transfer-Encoding: binary");
```

```
header("Content-Length: " . $this->filesize);
echo $this->certificate;
?>
<?php echo $this->render('header.phtml') ?>
<center>
<h1>Seu certificado foi renovado com sucesso.</h1>
<br/><br/>
<p> Clique <a href="../download/getcert">aqui</a> para fazer o download do seu novo certificado.</p>
</center>
<?php echo $this->render('footer.phtml') ?>

<?php echo $this->render('header.phtml') ?>
<center>
<h1>Seu certificado foi renovado com sucesso.</h1>
<br/><br/>
<p> Clique <a href="../download/getcert">aqui</a> para fazer o download do seu novo certificado.</p>
</center>
<?php echo $this->render('footer.phtml') ?>

<?php
header("Pragma: public");
# configura o tempo de expiracao da pagina
header("Expires: 0");
# faz com que o browser busque o arquivo no servidor e nao no cache
// header("Cache-Control: must-revalidate, post-check=0, pre-check=0");
header("Content-type: application/x-x509-user-cert");
// header("Content-Type: application/force-download");
// header("Content-Type: application/download");
// header("Content-Disposition: attachment; filename=" . $this->filename . ";");
header("Content-Transfer-Encoding: binary");
header("Content-Length: " . $this->filesize);
echo $this->certificate;
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
    <link rel="stylesheet" type="text/css" href="<?php echo $this->baseUrl ?>/styles/default.css" />
    <title><?php echo $this->escape($this->title); ?></title>
</head>
<body>
<div id="content">
<?php echo $this->render('header.phtml') ?>
```

```
<h2>Erro</h2>
<h3>Ocorreu um erro inesperado</h3>
<p>
<b>DescriÃ§Ã$o: </b><br/>
<pre><?php echo $this->escape($this->errorMessage) ?></pre>
</p>
<?php echo $this->render('footer.phtml') ?><?php echo $this->render('header.phtml') ?>
<h2>Erro</h2>
<h3>Ocorreu um erro inesperado</h3>
<p>
<b>DescriÃ§Ã$o: </b><br/>
<pre><?php echo $this->escape($this->errorMessage) ?></pre>
</p>
<?php echo $this->render('footer.phtml') ?><?php echo $this->render('header.phtml') ?>
<center>
<h1>Sistema de RenovaÃ§Ã$o On-line de Certificados Digitais</h1>
<br/><br/>
<a href="certificate/"> Renove aqui seu certificado digital</a>
</center>
<?php echo $this->render('footer.phtml') ?>
<?php echo $this->render('header.phtml') ?>
<center>
<h1>Sistema de RenovaÃ§Ã$o On-line de Certificados Digitais</h1>
<br/><br/>
<a href="certificate/"> Renove aqui seu certificado digital</a>
</center>
<?php echo $this->render('footer.phtml') ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
    <link rel="stylesheet" type="text/css" href="<?php echo $this->baseUrl ?>/styles/default.css" />
    <title><?php echo $this->escape($this->title); ?></title>
</head>
<body>
<div id="content">

</div>
</body>
</html>

<?php echo $this->render('header.phtml') ?>
<h2>Seu certificado foi renovado com sucesso.</h2>
```

```
<p> Clique <a href="../download/getcert">aqui</a> para fazer o download do seu novo certificado.</p>
<?php echo $this->render('footer.phtml') ?><?php echo $this->render('header.phtml') ?>

<h1>SubmissÃ$o de Arquivo</h1>
<h2>Selecione o arquivo contendo o certificado a ser renovado e clique em Enviar.</h2>
<p>obs: Um email serÃ¡ enviado para o endereÃ§o abaixo como prova de posse da chave privada.</p>
<form enctype="multipart/form-data" action="submitform" method="post">
<label for="mail">EndereÃ§o de Email:</label>
<input name="mail" type="text"/><br/>
<label for="mail">Arquivo contendo Certificado:</label>
<input name="upFile" type="file"/>
<input name="submit" type="submit" value="Enviar"/></td>
</form>
<?php echo $this->render('footer.phtml') ?>

<?php echo $this->render('header.phtml') ?>
<h1>Digite a resposta do desfio</h1>
<br/></br>
<h2>Um desafio foi enviado a voc\^{e} por email. Digite o c\'{o}digo recebido no email para proceder com a atu
<br/></br>
<form enctype="multipart/form-data" action="renewcert" method="post">
  <label for="challenge">Resposta:</label>
  <input name="challenge" type="text"/>
  <input name="submit" type="submit" value="Enviar"/>
</form>

<?php echo $this->render('footer.phtml') ?>

<?php echo $this->render('header.phtml') ?>
<h1>SubmissÃ$o de Arquivo</h1>
<h2>Selecione o arquivo contendo o certificado a ser renovado e clique em Enviar.</h2>
<p>obs: Um email serÃ¡ enviado para o endereÃ§o abaixo como prova de posse da chave privada.</p>
<form enctype="multipart/form-data" action="submitform" method="post">
<label for="mail">EndereÃ§o de Email:</label>
<input name="mail" type="text"/><br/>
<label for="mail">Arquivo contendo Certificado:</label>
<input name="upFile" type="file"/>
<input name="submit" type="submit" value="Enviar"/></td>
</form>
<?php echo $this->render('footer.phtml') ?>

<?php echo $this->render('header.phtml') ?>
<h2>Seu certificado foi renovado com sucesso.</h2>
<p> Clique <a href="../download/getcert">aqui</a> para fazer o download do seu novo certificado.</p>
```

```
<?php echo $this->render('footer.phtml') ?><?php echo $this->render('header.phtml') ?>
<h1>
<center>Selecione o mÃ©todo de renovaÃ§Ã$o</center>
</h1>
<br/><br/>
<center>
<h2>HÃ¡ duas formas de renovar seu certificado digital, escolha a forma mais adequada Ã s suas necessidades.</
<ol>
  <li><a href="renewssl" >Via conexÃ$o SSL</a>: VocÃª deverÃ¡ ter o certificado a ser renovado e sua chave priv
  <li><a href="showform" >Via upload de arquivo</a>: VocÃª selecionarÃ¡ o arquivo contendo o certificado no dis
</ol>
</center>
<?php echo $this->render('footer.phtml') ?>

<?php echo $this->render('header.phtml') ?>
<h1>Digite a resposta do desfio</h1>
<br/></br>
<h2>Um desafio foi enviado a voc\^{e} por email. Digite o c\'{o}digo recebido no email para proceder com a atu
<br/></br>
<form enctype="multipart/form-data" action="renewcert" method="post">
<label for="challenge">Resposta:</label>
  <input name="challenge" type="text"/>
  <input name="submit" type="submit" value="Enviar"/>
</form>
<?php echo $this->render('footer.phtml') ?>

<?php

function handle_uncaught_exception($ex) {
echo "<pre>" . get_class($ex) . ": " . $ex->getMessage() . " in line "
 . $ex->getLine() . " of file " . $ex->getFile() . "\n" .  $ex->getTraceAsString() . "</pre>";
}

set_exception_handler("handle_uncaught_exception");

error_reporting(E_ALL|E_STRICT);

date_default_timezone_set('America/Sao_Paulo');

set_include_path('.' . PATH_SEPARATOR . '../library/'
    . PATH_SEPARATOR . '../application/models/'
. PATH_SEPARATOR . '../application/utils/'
. PATH_SEPARATOR . '../application/exceptions/'
    . PATH_SEPARATOR . get_include_path());
```

```
include ’Zend/Loader.php’;
Zend_Loader::registerAutoload();

// setup controller
$frontController = Zend_Controller_Front::getInstance();
$frontController->throwExceptions(true);
$frontController->setControllerDirectory(’../application/controllers’);
// run!
$frontController->dispatch();
K 14
svn:executable
V 1
*
END
<?php

function handle_uncaught_exception($ex) {
echo "<pre>" . get_class($ex) . ": " . $ex->getMessage() . " in line "
 . $ex->getLine() . " of file " . $ex->getFile() . "\n" .  $ex->getTraceAsString() . "</pre>";
}

/**
* index.php
**/
set_exception_handler("handle_uncaught_exception");

error_reporting(E_ALL|E_STRICT);

date_default_timezone_set(’America/Sao_Paulo’);

set_include_path(’.’ . PATH_SEPARATOR . ’../library/’
    . PATH_SEPARATOR . ’../application/models/’
. PATH_SEPARATOR . ’../application/utils/’
. PATH_SEPARATOR . ’../application/exceptions/’
    . PATH_SEPARATOR . get_include_path());

include ’Zend/Loader.php’;
Zend_Loader::registerAutoload();

// setup controller
$frontController = Zend_Controller_Front::getInstance();
$frontController->throwExceptions(true);
$frontController->setControllerDirectory(’../application/controllers’);
```

```
// run!
$frontController->dispatch();
```