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

**Khristian Alexander Schönrock**

# Desenvolvimento de aplicações provedoras de serviços criptográficos: carimbos de tempo e certificados otimizados

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

Martín Augusto Gagliotti Vigil
Orientador

Prof. Ricardo Felipe Custódio, Dr.
Co-Orientador

Florianópolis, Julho de 2009

# Desenvolvimento de aplicações provedoras de serviços criptográficos: carimbos de tempo e certificados otimizados

Khristian Alexander Schönrock

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

---

Prof. Luis Fernando Friedrich, Ph.D.

Coordenador do Curso

Banca Examinadora

---

Martín Augusto Gagliotti Vigil

Orientador

---

Prof. Ricardo Felipe Custódio, Dr.

Co-Orientador

---

Roberto Samarone dos Santos Araújo, Ph.D.

---

Juliano Romani, M.Sc.

---

Marcelo Carlomagno Carlos, M.Sc.

*"Far away there in the sunshine are my aspirations. I may not reach them, but I can look up and see their beauty, believe in them and try to follow where they may lead."*
*Louisa May Alcott*

Ofereço este trabalho à minha mãe e meu pai, por terem me ajudado (e me sustentado) durante todo o curso, e também ao Odin por (quase) sempre vir me receber quando voltava para casa.

# Agradecimentos

Em primeiro lugar, agradeço à minha família por ter me incentivado a continuar estudando e a acreditar em mim mesmo, e nas escolhas que faço.
Também agradeço ao professor Ricardo Felipe Custódio pela oportunidade de integrar a equipe do LabSEC e de realizar este trabalho, especialmente Juliano Romani e Martín Augusto Gagliotti Vigil por terem me orientado neste trabalho.
Finalmente, agradeço a toda a equipe do LabSEC por fazerem estes dois semestres de trabalho valerem a pena, e pela enorme paciência para me ajudar quando eu tinha dúvidas.

# Sumário

# Lista de Figuras

# Lista de Siglas

| | |
|---|---|
| **AC** | Autoridade Certificadora |
| **API** | Application Programming Interface |
| **CBC** | Cipher Block Chaining |
| **DES** | Data Encryption Standard |
| **ICP** | Infra-estrutura de Chave Pública |
| **LabSEC** | Laboratório de Segurança em Computação - UFSC |
| **CRL** | *Certificate Revocation List* - Lista de Certificados Revogados |
| **PHP** | Hypertext Preprocessor |
| **RSA** | Rivest-Shamir-Adleman |
| **SHA** | Secure Hash Algorith |
| **SQL** | Structured Query Language |
| **ACT** | Autoridade Certificadora Temporal |
| **SCT** | Servidor de Carimbo de Tempo |
| **OC ou CO** | Certificado Otimizado |
| **ACCO** | Autoridade Certificadora de Certificados Otimizados |
| **SSL** | Secure Socket Layer |
| **MRS** | Módulo de Relógio Seguro |
| **HSM** | Hardware Security Module |
| **ASN.1** | Abstract Syntax Notation One |
| **OID** | Object Identifier |
| **HTTP** | Hypertext Transfer Protocol |
| **WSDL** | Web Service Definition Language |

# Resumo

Trabalhos acadêmicos, principalmente de pesquisa, normalmente necessitam de meios para verificar como os novos produtos, algoritmos ou procedimentos se comportam quando usados com mecanismos já existentes e/ou de uso corrente. O propósito deste trabalho é a implementação de aplicações que forneçam serviços criptográficos para uso esporádico ou mesmo para serem integrados em outras aplicações. Este trabalho documenta a implementação de um Servidor de Carimbo de Tempo (SCT) e um conjunto de programas para suporte à emissão de certificados otimizados.

Palavras-chave: carimbo de tempo, certificado otimizado, ICP

# Abstract

Academic works, specially ones involving original research, usually need some means for verifying how their new products, algorithms or procedures behave when used with already existant and/or currently employed mechanisms. The purpose of this work is to implement applications to provide cryptographic services for sporadic use or even for use embedded in other applications developed later. This work documents the implementation of a Time Stamp Server (TSS) and a set of applications meant to support the emission of optimized certificates.

Keywords: time stamp, optimized certificate, PKI

# Capítulo 1

# Introdução

Com o uso crescente do comércio eletrônico, normalmente usando meios inseguros como a internet, percebe-se a necessidade cada vez maior da criação de processos que permitam assegurar um nível satisfatório de segurança nas transações. Embora mecanismos de segurança já estejam razoavelmente desenvolvidos para uso no comércio eletrônico, novos produtos continuam a aparecer, como o uso pelo governo de processos licitatórios totalmente eletrônicos. O uso de documentos eletrônicos deve conter alguns requisitos de segurança: autenticidade, integridade, irretratabilidade, e confidencialidade. Juntamente a estes requisitos deve-se utilizar evidências temporais.

Segundo Custódio et al [1], autenticação e integridade são alcançados com o uso de métodos criptográficos como resumos criptográficos em associação com esquemas de assinatura digital. A irretratabilidade pode ser considerada tecnologicamente alcançada através do uso de autenticação. Confidencialidade é alcançada com o uso de algoritmos de criptografia simétrica. A evidência temporal é o preciso instante a partir do qual se pode provar a existência de um documento eletrônico, juntamente com o fato de que o documento não foi alterado desde então. Esta evidência temporal é fornecida por uma terceira parte confiável, uma entidade chamada Autoridade de Carimbo de Tempo (ACT), que emite um carimbo de tempo. Um exemplo de uso da evidência temporal em conjunto com os outros requisitos de segurança é uma licitação eletrônica, onde a informação é entregue antes de uma data limite, e só

pode ser lida a partir de uma outra data pré-definida, devendo os dados permanecerem em segredo no período entre as duas datas. Para a implementação de tais objetivos, Loren M. Kohnfelder propôs um esquema de certificados [2] que, aliado ao uso de criptografia assimétrica, resultou num sistema hoje conhecido como infra-estrutura de chaves públicas - ICP.

Em uma ICP, validar uma assinatura é uma tarefa normalmente custosa em termos de utilização de recursos de rede e processamento. A utilização da rede depende de quanta informação o validador ainda precisa obter, O usuário precisa verificar se o certificado é válido (construindo o caminho de certificação), checar se o certificado não expirou, se não foi revogado, se a assinatura é válida, entre outros. Neste processo são utilizadas informações assinadas (como as CRLs), que também precisam ser verificadas. Esta solução, porém, não é viável em todos os ambientes: em máquinas com pouco poder de processamento ou memória (normalmente ambos), a construção de caminhos de certificação e validação de certificados pode facilmente ultrapassar as capacidades de armazenamento do dispositivo [3]. Uma proposta para contornar este problema e simplificar a validação de certificados é o uso de certificados otimizados [4, 5], que eliminam o uso de CRLs aplicando o conceito de certificados de curta duração e facilitam a verificação do status de um certificado usando o esquema de provas Novomodo [6].

## 1.1 Justificativa

O Servidor de Carimbo de Tempo foi desenvolvido como parte da implementação do sistema idealizado por Juliano Romani em seu trabalho “Integração de Serviços de Relógio para Infra-estrutura de Chaves Públicas”[7]. A segunda parte do trabalho se baseia na pesquisa de Martín Vigil [4] e trata dos certificados otimizados e a ICP que lhes dá suporte. Como são resultantes de pesquisa recente, existe a necessidade de verificar seu funcionamento e sua viabilidade de uso em conjunto com as ferramentas e algoritmos já existentes. Com isso, poder-se-á verificar, por exemplo,

a aderência das especificações das novas extensões e mudanças feitas nos certificados às normas (e suas implementações) vigentes. O desenvolvimento destas aplicações faz-se útil no âmbito acadêmico pois podem ser usadas por outros projetos, evitando que os desenvolvedores tenham de implementar novamente as ferramentas que usarão para testar seus programas. A solução fornecida também é gratuita e leve, assim o desenvolvedor não necessita buscar ferramentas externas e possivelmente sujeitas a restrições de licenças.

## 1.2 Objetivos

### 1.2.1 Objetivos Gerais

Desenvolver aplicações de emissão de carimbos de tempo e de suporte à emissão de certificados otimizados, à criação de provas de Novomodo e o gerenciamento das provas geradas, e à consulta às provas atuais de cada certificado de ACCO (Autoridade Certificadora de Certificados Otimizados) emitido.

### 1.2.2 Objetivos Específicos

- Desenvolver um SCT (Servidor de Carimbo de Tempo) utilizando a biblioteca libcryptosec-tsp;

- Integrar o SCT ao HSM (Hardware Security Module);

- Criar uma versão do SCT para execução em modo *daemon* no sistema operacional FreeBSD;

- Testar a viabilidade do uso dos certificados otimizados em conjunto com sistemas já existentes.

## 1.3 Trabalhos Relacionados

Os trabalhos de Juliano Romani [7] sobre Autoridades de Carimbo de Tempo e Servidores de Carimbo de Tempo serviram de base para o projeto do Servidor de Carimbo de Tempo implementado na primeira etapa deste trabalho.

Nestes trabalhos é descrito um Módulo de relógio Seguro e a sua integração em uma ICP. Um Módulo de Relógio Seguro é um HSM que integra os serviços de tempo, garantindo maior confiabilidade e segurança nas operações criptográficas realizadas, além de permitir que outras aplicações façam uso destas melhorias.

Martín Vigil [4, 5] menciona em seus trabalhos possíveis problemas nos processos de validação de cadeias de certificados, por serem operações com elevados custos de processamento computacional, uso de rede e (potencialmente) uso de espaço em memória tanto volátil como não-volátil. Para resolver estas questões ele propõe o uso de certificados otimizados em conjunto com provas de novomodo para amenizar os problemas supracitados.

## 1.4 Metodologia

A criação deste trabalho seguiu a metodologia da pesquisa bibliográfica, por ter sido baseado em trabalhos já existentes. As principais fontes foram trabalhos acadêmicos e artigos, sendo usados também textos encontrados na web, livros sobre criptografia e documentos técnicos.

O trabalho iniciou-se com o estudo dos trabalhos usados como ponto de partida e motivação para este: os trabalhos de Juliano Romani [7, 5] foi a base para o estudo e desenvolvimento do primeiro objetivo, de desenvolver um servidor de carimbo de tempo; o trabalho de Martín Vigil [4, 5] foi usado como base para o estudo de ICP de certificados otimizados e o projeto das aplicações necessárias para os testes e o funcionamento desta ICP. Após o estudo destes trabalhos e discussão dos assuntos a serem cobertos por este trabalho, chegou-se aos requisitos e objetivos especificados em 1.2.

Para verificação dos resultados obtidos na primeira parte do trabalho, foram feitos testes usando o *software* Adobe Acrobat 9. Para os testes das aplicações relacionadas aos certificados otimizados, foram utilizados *softwares* como o OpenSSL para validação das adaptações feitas aos certificados.

Estas decisões fizeram necessário também o estudo de várias ferramentas e tecnologias, como a linguagem C++ [8], a linguagem PHP [9], a biblioteca LibCryptoSec [10] e demais tecnologias listadas na seção 5.

## 1.5 Estrutura do Trabalho

No decorrer deste trabalho são descritos os aspectos teóricos e práticos da implementação dos programas criados, começando com uma recapitulação dos conceitos básicos sobre criptografia necessários para a compreensão deste trabalho (capítulo 2), passando à implementação do servidor de carimbo de tempo (capítulo 3) e, depois, à implementação das aplicações relacionadas à emissão de certificados otimizados (capítulo 4). Após o desenvolvimento, será feita uma breve explanação das ferramentas utilizadas ao longo dos trabalhos (capítulo 5), seguida pelas conclusões alcançadas.

# Capítulo 2

# Conceitos

O termo *criptografia* significa escrita oculta ou secreta (do grego (*kryptos*, "oculto, secreto"; e (*gráphō*), "escrita"), e é geralmente visto como o processo de ocultar uma informação qualquer ou de tornar legível uma informação oculta. Este processo de tornar a informação ilegível a terceiros provê a *confidencialidade* da informação. Técnicas modernas de criptografia permitem assegurar outras propriedades de informações [11]:

- *integridade*, ou a certeza de que a informação não foi alterada desde que deixou as mãos do seu emissor;
- *autenticação*, ou o conhecimento da autoria da informação;
- *irrefutabilidade*, ou a impossibilidade de alguém negar algo que tenha feito (como o envio de uma mensagem, por exemplo).

Ao processo de ocultar uma informação usando criptografia dá-se o nome de cifragem, enquanto que ao processo contrário, de tornar a informação novamente inteligível, dá-se o nome de decifragem [11]. Antes do surgimento de computadores estes processos eram normalmente manuais, e a segurança da informação normalmente era ligada à capacidade de se manterem ocultos os algoritmos usados para cifrar a informação. Como os computadores tornaram mais fácil o processo de reverter a cifragem destes algoritmos primitivos, foi necessário desenvolver novos algoritmos criptográficos. Existem

vários algoritmos criptográficos, mas a maioria deles trabalha com duas entradas: uma informação original e um valor secreto, chamado de chave [12].

Já o processo de recuperar a informação original de sua forma cifrada sem todos os dados necessários para aplicar o algoritmo de decifragem é chamada de criptoanálise (uma tentativa de criptoanálise é normalmente chamada de ataque). Criptoanálise pode ser feita de várias formas: ataque à chave, ataque ao algoritmo, entre outras.

## 2.1 Criptografia Simétrica

Na criptografia simétrica, a mesma chave é usada para cifrar e decifrar a informação. Assim, o emissor deve compartilhar a chave usada na cifragem com o receptor[1] [12].

Sistemas de criptografia simétrica são bastante eficientes computacionalmente falando, sendo seu uso recomendado quando há a necessidade de cifrar grandes quantidades de informação. Para que um atacante possa obter a informação cifrada, ele deve adivinhar a chave ou interceptá-la. Para evitar que o atacante adivinhe a chave, ela deve ser composta de bits gerados o mais aleatoriamente possível. Para evitar que o atacante intercepte a chave, ela deve ser cifrada durante o envio.

Atualmente, o DES - *Data Encryption Standard* e o AES - *Advanced Encryption Standard*[13] são os algoritmos de criptografia simétrica mais utilizados.

## 2.2 Criptografia Assimétrica

Criptografia assimétrica também pode ser chamada de criptografia de chaves públicas, por haver uma chave que deve ser mantida secreta (chave privada) e outra que pode ser publicada (chave pública). As duas chaves são complementares, mas o valor da chave privada não pode ser derivado da chave pública. Criptografia assimétrica

[1] Por isso, sistemas que usam criptografia simétrica são também chamados de sistemas de chave secreta compartilhada.

simplifica sobremaneira o gerenciamento e troca de chaves para criptografia simétrica [12].

Criptografia assimétrica requer, entretanto, muito mais poder computacional que operações equivalentes usando criptografia simétrica. Por isso não é normalmente usada para cifragem de dados. Ao invés disso, a criptografia assimétrica é usada para auxiliar no compartilhamento seguro de uma chave simétrica, que é então usada para cifrar os dados. Há dois mecanismos deste tipo: acordo de chave (como exemplo, o algoritmo de Diffe-Hellman [14] ) e transporte de chave (como exemplo, o algoritmo RSA) [12].

Algoritmos de criptografia assimétrica necessitam de chaves muito maiores do que algoritmos de criptografia simétrica de força equivalente. A segurança das chaves na criptografia assimétrica é garantida pela dificuldade de se fatorar números primos muito grandes. Mas isso por si só não é suficiente para garantir a segurança destes algoritmos por muito tempo: o tempo todo são desenvolvidas novas técnicas de fatoração e algoritmos mais eficientes, além de o poder de processamento das máquinas aumentar cada vez mais [11].

## 2.3 Resumo criptográfico

Funções de resumo criptográfico, também chamadas de funções de *hash*, são funções que tomam como entrada um número arbitrário de bits e produzem uma saída de tamanho fixo. Embora muitas funções tenham esta característica, funções de *hash* devem ter ainda duas características adicionais [12]:

- deve ser computacionalmente impossível recuperar os dados de entrada a partir do *hash* gerado;
- deve ser computacionalmente impossível gerar duas entradas diferentes que produzam o mesmo valor quando passadas pela função de hash.

Estas funções são essenciais na criptografia moderna, aparecendo como bloco básico de muitos protocolos criptográficos. A principal fonte de segurança da

função de *hash* está na impossibilidade (computacional) de realizar a operação inversa. A mudança de um único bit na entrada muda, em média, metade dos bits gerados na saída [11].

Como algoritmos de *hash* populares atualmente, pode-se citar os da família SHA2 (com saída de tamanho variado, normalmente de 224 a 512 bits) e RIPE-MD (com saída de 128 bits).

# Capítulo 3

# Servidor de Carimbo de Tempo

## 3.1 Carimbo de Tempo

### 3.1.1 Introdução

O carimbo de tempo foi criado a partir da necessidade de adicionar às assinaturas digitais uma informação que atuasse como âncora temporal para estes dados. A especificação do formato dos carimbos de tempo e do protocolo de comunicação entre clientes e servidores é dada pela norma RFC 3161 [15]. Um carimbo de tempo é composto pela identificação dos dados, normalmente um *hash* dos dados a serem assinados, e a data e hora em que o carimbo foi gerado. Estes dados são então enviados pelo requisitor à ACT, assinados com a chave privada da ACT e enviados de volta ao requisitor, que deve checar (a) se o que foi carimbado corresponde ao que foi enviado ao servidor de carimbo de tempo (SCT), (b) se o certificado pertence à ACT, e (c) checar o tempo incluso na resposta com uma fonte confiável de tempo. Adicionalmente, pode-se incorporar à requisição um número aleatório (chamado de *nonce* - *number once*), e verificando depois se o nonce recebido na resposta corresponde ao que foi enviado. A adição deste *nonce* também ajuda na prevenção contra ataques de repetição (onde um atacante usa mensagens gravadas previamente para tentar subverter o protocolo de comunicação ou autenticação [11]).

Como o que é enviado ao SCT é apenas o *hash* dos dados, o carimbo de tempo pode ser usado para carimbar documentos confidenciais (o documento original só será necessário na hora de verificar a sua integridade). O carimbo também pode ser renovado, de modo a ser válido por tanto tempo quanto necessário (segundo método apresentado em [16]).

### 3.1.2 Usos do carimbo de tempo

A seguir, exemplos de situações onde um carimbo de tempo é usado:

1. Prova de que um documento existia a partir de um instante no tempo: uso básico do carimbo de tempo, onde a ACT funciona como terceira parte confiável.

2. Base para serviços de irretratabilidade: um serviço de irretratabilidade necessita de uma âncora temporal para complementar as assinaturas (como exemplo, o próximo item).

3. Checagem de assinaturas: uma chave pode pertencer a uma pessoa apenas durante um tempo definido, como a duração de um emprego, ou a chave pode ser comprometida, e assim não ser mais adequada para uso em assinaturas. Quando a pessoa não mais detém a chave, é emitido um certificado de revogação da chave, e isso não invalida as assinaturas feitas anteriormente. Assim, se for necessário checar a assinatura de algum documento assinado com uma chave revogada, o carimbo de tempo é usado para determinar se o documento foi assinado enquanto a chave ainda era válida. Isso também evita que um signatário que falsamente comunique o comprometimento da sua chave coloque maliciosamente em questão uma assinatura sua em um documento qualquer.

4. Envio de documentos com data-limite para entrega: quando há necessidade de entregar um documento até uma certa data-limite, o carimbo de tempo pode ser usado para indicar a hora em que um documento foi enviado ou submetido.

## 3.2 Autoridade de Carimbo de Tempo

Uma ACT é uma entidade que atua como terceira parte confiável, assinando carimbos de tempo emitidos pelo SCT para comprovar a existência de um dado em um determinado instante no tempo. Segundo a RFC3161 ([15]), são obrigações da ACT:

1. usar uma fonte de tempo confiável;
2. incluir um valor de tempo confiável em cada carimbo de tempo gerado;
3. gerar um número inteiro único para cada novo carimbo de tempo gerado;
4. produzir um carimbo de tempo quando receber uma requisição válida de um cliente, se possível;
5. incluir no carimbo de tempo um identificador que indique unicamente a política de segurança sob a qual o carimbo foi gerado;
6. somente carimbar uma representação em resumo criptográfico (hash) dos dados;
7. examinar o OID do algoritmo de *hash* e verificar se o comprimento do *hash* é consistente com o algoritmo;
8. não examinar o *hash* sendo carimbado em forma alguma (exceto para determinar seu comprimento, como especificado no item anterior);
9. não incluir qualquer informação do requisitor do carimbo no carimbo de tempo gerado;
10. assinar cada carimbo com uma chave gerada especificamente para este propósito, e explicitar esta propriedade no certificado correspondente;
11. incluir informações adicionais no carimbo de tempo gerado, se solicitado pelo requisitor usando o campo de extensões, somente para extensões suportadas pela ACT. Se não for possível, a ACT deve responder com um erro.

## 3.3 Servidor de Carimbo de Tempo

Um servidor de carimbo de tempo é o responsável por receber as requisições de carimbos de tempo encaminhadas à ACT, gerar os carimbos e devolvê-los à ACT, que se encarrega de enviar as respostas de volta aos remetentes. Neste trabalho foi desenvolvido um SCT, com o objetivo de integrá-lo a um Módulo de Relógio Seguro (MRS), como especificado por [7]. O servidor foi concebido para responder a uma requisição por vez (ou seja, carimbos de tempo não serão gerados em paralelo) e rodar em modo daemon (desacoplado da linha de comando).

### 3.3.1 Implementação

Este SCT foi desenvolvido usando como base a biblioteca criptográfica com suporte a funções de carimbo de tempo desenvolvida no LabSEC/UFSC, a *libcryptosec-tsp*[10], e rodando em ambiente Linux[17]. Mais tarde foi feita uma versão para o ambiente *FreeBSD*.

Para comunicação com os clientes, o servidor usa o protocolo *HTTP 1.1*, conforme especificação da RFC3161. O SCT recebe as requisições *HTTP* do cliente contendo uma requisição de carimbo de tempo (*timestamp-request*) em formato *DER*, e usa o relógio do sistema operacional para gerar o carimbo de tempo e enviar de volta a resposta (*timestamp-response*). A chave privada usada na assinatura e o certificado ficam guardados no *HSM*, sendo recuperados pelo SCT no momento da geração do carimbo.

Para evitar que uma requisição mal-formada cause um erro fatal no servidor, o servidor foi modificado para passar cada requisição recebida a um novo processo, de modo a isolar a geração do carimbo de tempo. Para manter o pré-requisito anterior, de gerar um carimbo por vez, foram usados semáforos de processo (*System V semaphores*) para sincronizar os processos criados.

### 3.3.2 Testes

O SCT foi testado em ambiente Linux e posteriormente no *FreeBSD*. Para geração das requisições foi usado o software *Adobe Acrobat Pro Extended 9*, que possui funções para assinar documentos em formato *PDF* gerados pelos usuários. Testes usando o *HSM* não foram possíveis até o momento por conta de uma incompatibilidade da *libcryptosec-tsp* em sua versão atual com a versão do *OpenSSL* usado pelo *HSM*.

Após concluídos os primeiros testes e, confirmando o funcionamento do servidor, foi implementado o modo *daemon* para desacoplá-lo da linha de comando e permitir que o servidor pudesse ser inicializado quando o sistema for carregado.

### 3.3.3 Resultados

O SCT funcionou como esperado, respondendo às requisições rapidamente (por rapidamente entenda-se rápido o suficiente para que a aplicação de teste não tivesse de fazer nova requisição após um período de *timeout*), e os carimbos foram gerados conforme a especificação, sendo reconhecidos sem problema na aplicação de teste. Excetuando-se a integração com o HSM, os objetivos específicos relacionados ao SCT foram atingidos.

# Capítulo 4

# Emissão de Certificados Otimizados

Os certificados otimizados foram concebidos com o objetivo de reduzir os requerimentos de espaço (tanto em memória volátil quanto em memória não-volátil) de certificados normais e reduzir os requerimentos de banda de rede e processamento computacional para a construção e validação de caminhos de certificação. Além disso, certificados otimizados podem substituir carimbos de tempo [5].

## 4.1 Certificado Otimizado

Certificados otimizados têm como objetivo diminuir os custos da validação de certificados usados para assinar documentos eletrônicos [5]. Um certificado otimizado é emitido para um documento específico, funcionando também como carimbo de tempo para este documento, no sentido de atestar que a informação assinada existia no momento em que o certificado foi emitido.

Para evitar o uso de CRLs, os certificados otimizados extendem o princípio dos certificados de curta duração, fazendo os campos *notBefore* e *notAfter* serem iguais. Como o certificado era válido apenas no instante de sua emissão, não é necessário revogá-lo. O certificado otimizado será válido enquanto o certificado do seu emissor (a ACCO) for válido. O certificado otimizado é um certificado X.509 v3 acrescido de 3 extensões novas:

- *novomodo-validity-proof*: a prova de Novomodo na hora da emissão do certificado;

- *SignedDocDigest*: o *hash* do documento assinado;

- *ValidationTime*: o instante da validação da assinatura do documento.

Para a fonte do horário usado na extensão *ValidationTime* existem certos critérios que definem se ela é confiável ou não. Estes critérios estão resumidos na tabela 4.1:

**Tabela 4.1:** Referência de tempo para validação da assinatura

| Fonte | Descrição | Confiável |
|---|---|---|
| Momento da requisição do CO (presente) | Relógio interno da ACCO | Sim |
| Passado | Um carimbo de tempo da assinatura do documento | Sim |
| Passado | Um CO prévio | Sim |
| Passado | Assinante do documento | Não |
| Passado | Verificador de documento | Não |

Da tabela 4.1 vê-se que datas provenientes de fontes confiáveis (relógio da ACCO, carimbo de tempo emitido por uma ACT, um certificado otimizado emitido previamente usando uma fonte de tempo confiável) poderão ser usadas para especificar a data em que a assinatura era válida. Datas no passado e não provenientes de uma fonte confiável não poderão ser usadas.

A extensão *novomodo-validity-proof* recebe a prova de Novomodo atual, e a extensão *SignedDocDigest* recebe o *hash* do documento sendo assinado. A estrutura básica de um certificado otimizado é mostrada na figura 4.1.

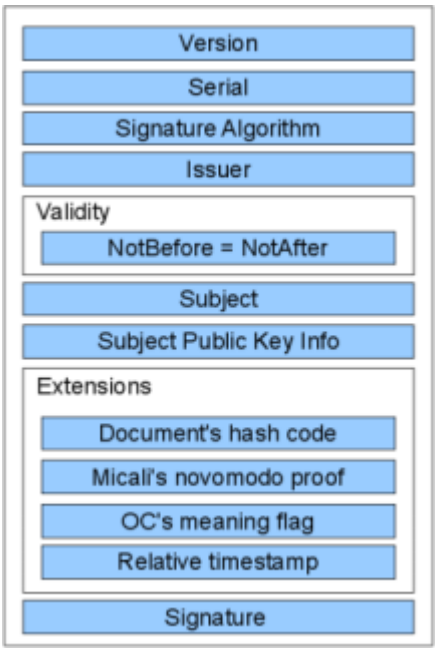


**Figura 4.1:** Estrutura de um Certificado Otimizado.

## 4.2 AC de Certificados Otimizados

Uma Autoridade Certificadora de Certificados Otimizados (ACCO) funciona como uma AC normal, para certificados otimizados. Ela é responsável por receber as requisições dos usuários, validá-las e enviar os resultados de volta aos remetentes. Embora não seja necessário checar a revogação do certificado otimizado, ainda é necessário verificar se o certificado da ACCO ainda está válido. Para isso é empregado o esquema Novomodo [6], que substitui a construção e validação da cadeia de certificados e checagem de CRLs por comparação de hashes, diminuindo bastante os recursos computacionais necessários para a validação do certificado otimizado.

O certificado da ACCO tem uma extensão adicional: *NovomodoTargets*, que contém o OID do algoritmo de *hash* usado, o alvo de validade, o algo de revogação e a granularidade.

## 4.3 Emissão de Certificados Otimizados

Para otimizar um certificado existente, o fluxo segue como exposto abaixo e na figura (4.3).

1. O cliente reúne os seguintes itens:
    - Seu certificado;
    - O documento assinado;
    - Todos os certificados que compõe a cadeia de certificação do seu certificado;
    - As CRLs necessárias;
    - Carimbos de tempo (se houver);
    - O *hash* do documento assinado (para verificação).
2. O servidor checa a validade o certificado do usuário e a assinatura do documento;
3. Se válidos, obtém a prova novomodo atual para o certificado da ACCO, e gera o certificado otimizado;
4. Retorna o certificado otimizado para o cliente.

## 4.4 Implementação

Para suportar a emissão de certificados otimizados, foi necessária a implementação de dois componentes separados:

- Cliente/Servidor para uso do Web Service de emissão de certificados otimizados;
- Sistema para gerar, gerenciar e consultar provas de Novomodo necessárias para a emissão dos certificados da ACCO e para checagem de validade dos certificados otimizados emitidos.

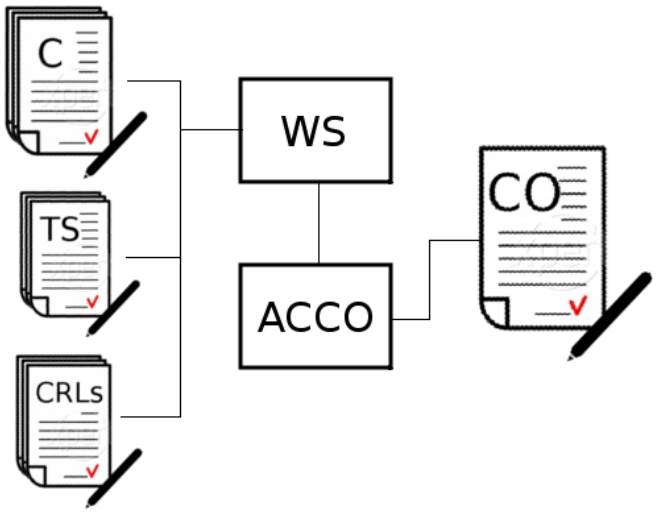


**Figura 4.2:** Passos da emissão de um certificado otimizado.

### 4.4.1 Web Service para Emissão de Certificados Otimizados

Para a emissão de certificados otimizados, foi projetado um sistema disponível online via Web Service, a fim de possibilitar o acesso ao serviço sem a necessidade de se preocupar com a forma de implementação do cliente (Java, C++, etc). Para isso, foi utilizado o framework Apache Axis2 ([18]), em sua versão para Java, para fornecer a base para o Web Service. O web service em si foi implementado também em Java, usando a API criptográfica fornecida pelo BouncyCastle ([19]).

Para o envio dos dados ao servidor são disponibilizadas dois métodos: um que recebe um CMS contendo todos os dados requeridos (ver lista 4.3), e outro que recebe os dados separadamente (hash, certificado do usuário, certificados intermediários, CRLs e carimbos de tempo). Nos dois modos, os dados são codificados em Base64 para transporte. A codificação em Base64 facilita a interoperabilidade entre implementações diferentes do serviço.

O servidor, ao receber os dados, verifica se o certificado do usuário é válido (tentando construir seu caminho de certificação), se a assinatura no CMS é válida e, finalmente, se o *hash* assinado é igual ao *hash* que foi enviado para verificação. Sendo os dados válidos, o servidor passa então à emissão do certificado, obtendo a

prova de Novomodo atual para o certificado da ACCO emissora, e inserindo no certificado X.509 resultante as extensões específicas dos certificados otimizados: *novomodo-validity-proof* contendo a prova recém-obtida, *SignedDocDigest* contendo o *hash* do documento assinado, e *ValidationTime*.

Como o Apache Axis2 permite que classes Java comuns sejam usadas como web services, as classes responsáveis pela emissão do certificado otimizado puderam ser implementadas de modo a permitir seu uso em outras aplicações Java, sem a necessidade de importar também a função de web service.

### 4.4.2 Geração e Gerenciamento de Provas Novomodo

O gerenciamento das provas de Novomodo foi implementado em uma aplicação separada, e feita como uma aplicação web para manipulação direta pelos usuários. Esta aplicação foi implementada em linguagem PHP, rodando no servidor HTTP Apache 2 e usando um banco de dados MySQL para persistência, e foi implementada para facilitar o gerenciamento de provas Novomodo utilizadas nos testes realizados.

A aplicação foi dividida em duas telas: uma tela para o cadastro de novas provas, e outra para listagem de provas geradas, consulta a provas individuais, consulta de provas usando uma data informada pelo usuário, e a remoção de provas geradas.

Na tela de cadastro de provas (ver figura 4.4.2, o usuário deve entrar com uma identificação (nome do emissor do certificado), o número de série do certificado, as datas de emissão e validade (*notBefore* e *notAfter*), e a granularidade. As datas deverão ser entradas em formato GENERALIZEDTIME (parte do padrão ASN.1 [20]), devendo incluir todos os dígitos até os segundos no formato YYYYMMDDHHSSZ: 4 dígitos para ano, 2 dígitos para mês (1-12), 2 dígitos para dia (1-31), 2 dígitos para hora (0-23), 2 dígitos para minutos (0-59) e 2 dígitos para segundos (0-59), seguidos da letra “Z” (de “Zulu Time”).

Ao salvar os dados, a aplicação gerará 2 números aleatórios secretos, $X_0$ e $Y_0$. A partir das datas de emissão e validade e da granularidade, será calculado o valor

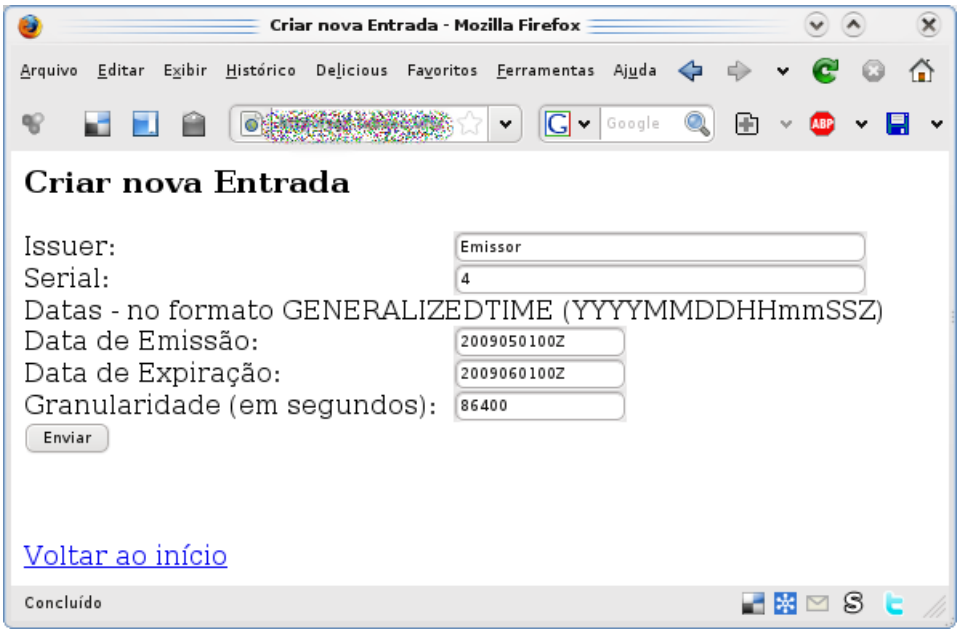


**Figura 4.3:** Tela de cadastro da aplicação de gerenciamento de provas de Novomodo.

$n$, o número de provas geradas. Então, aplicando a função de *hash* (nesta implementação foi usado o algoritmo de *hash* SHA1) ao número secreto $Y_0$ será obtido o alvo de revogação ($Y_1$). Ao aplicar a função de *hash* $n$ vezes ao número $X_0$, será obtido o valor $X_n$ (alvo de validade). Destes, serão salvos as datas de emissão e validade, os números secretos $X_0$ e $Y_0$, a granularidade e as informações de emissor e número de série. Assim, sempre que necessário executar uma consulta a uma prova, as provas podem ser regeradas a partir dos dados armazenados.

Na tela de consulta de provas (ver figura 4.4.2, o usuário pode selecionar uma das provas já cadastradas na lista *drop-down* e, ao clicar no botão exibir, serão exibidas todas as informações geradas pela aplicação no ato do cadastro.

Ao carregar uma prova, a aplicação gerará novamente o número total de provas ($n$), as provas ($Y_1$, $X_1$ - $X_n$), e exibirá, além dos dados armazenados, o período atual ($k$), a prova do período atual ($X_{n-k}$) e os alvos de validade e revogação. Enquanto exibe uma prova previamente armazenada, o usuário pode consultar a prova de Novomodo em um instante específico (passado ao sistema em formato GENERALIZEDTIME).

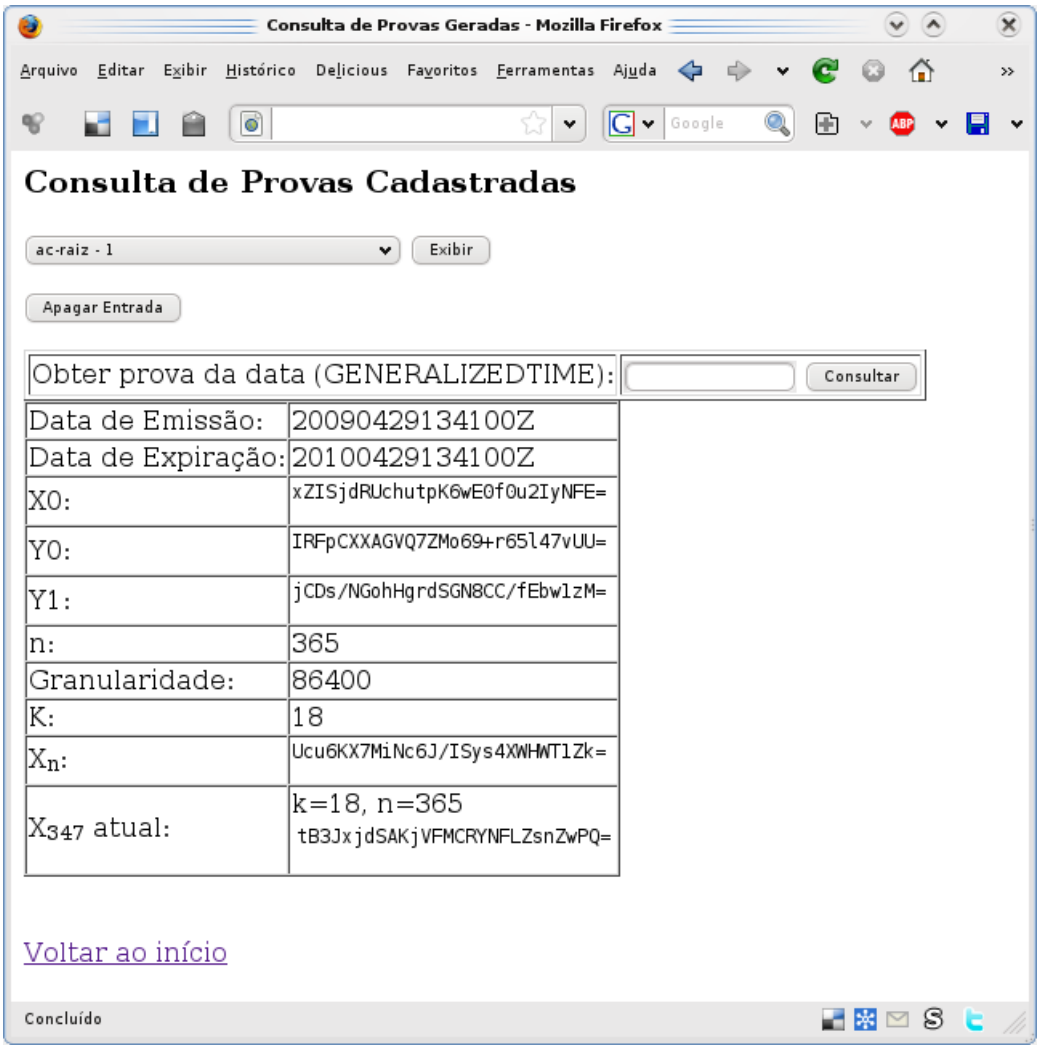


**Figura 4.4:** Tela de consulta da aplicação de gerenciamento de provas de Novomodo.

### 4.4.3 Resultados

A implementação das especificações dos certificados otimizados (adição das extensões, adaptações) permitiu que fossem feitas alterações e melhorias às mesmas, e foi possível integrar os certificados otimizados gerados com produtos existentes (por exemplo, *OpenSSL*).

# Capítulo 5

# Tecnologias Utilizadas

As tecnologias utilizadas neste trabalho foram escolhidas principalmente por serem de código aberto e/ou gratuitas. Outros critérios foram a familiaridade com elas e compatibilidade com outras tecnologias já sendo utilizadas (linguagens de programação, bibliotecas, entre outros).

## 5.1 Apache Axis2

O Apache Axis2 [18] é uma implementação da especificação de SOAP submetida à W3C. Da especificação (W3C, 2000 [21]):

> SOAP é um protocolo leve para troca de informações estruturadas num ambiente distribuído. É um protocolo baseado em XML que consiste de três partes: um envelope que define a estrutura para descrever a informação contida na mensagem e como processá-la, um conjunto de regras de codificação para expressar instâncias de tipos de dados definidos por aplicações, e uma convenção para representar chamadas de procedimento remoto e respostas.

O Axis2 foi útil neste trabalho por fornecer um mecanismo simples para expor os métodos de uma classe Java como Web Services, e por ser possível automatizar a geração de clientes para estes Web Services usando seus descritores WSDL.

## 5.2 Bouncy Castle

O Bouncy Castle[19] é uma api de serviços criptográficos para Java e C#, oferecendo um provedor para a Java Cryptography Extension e Java Cryptography Architecture, além de ferramentas para lidar com certificados X.509, carimbos de tempo, e outros serviços.

O Bouncy Castle foi útil neste trabalho para a manipulação de certificados e tarefas relacionadas a ICP (emissão de certificados, construção e validação de caminho de certificação, por exemplo).

## 5.3 OpenSSL

O OpenSSL [22] é um conjunto de ferramentas criptográficas de código aberto e gratuito, implementando os protocolos SSL v2/v3, TLS v1 e uma biblioteca de funções criptográficas de uso geral.

Além de ter sido utilizado como meio de verificar a validade dos certificados gerados pela aplicação de emissão de certificados otimizados, é indiretamente usado através da LibCryptoSec[10].

## 5.4 LibCryptoSec

A LibCryptoSec[10] é uma abstração do conjunto de funções criptográficas OpenSSL[22], disponibilizando suas funções para uso em programação orientada a objetos. Implementada em C++, seu objetivo é flexibilizar e facilitar o desenvolvimento de soluções criptográficas ao evitar que desenvolvedores necessitem aprender a API do *OpenSSL*.

## 5.5 PHP

PHP é uma linguagem de programação interpretada voltada para uso na web, podendo ser misturada ao código HTML existente. Pode ser usado como linguagem de scripting no servidor (*server-side*), diretamente na linha de comando e, embora não seja o seu forte, como linguagem de programação para aplicações desktop.

## 5.6 Linux

O Linux[17] é um sistema operacional gratuito e de código aberto desenvolvido pelo engenheiro de software finlandês Linus Torvalds desde 1991. Hoje é mantido e melhorado por uma comunidade internacional de desenvolvedores aberta. Todo o desenvolvimento deste trabalho foi feito em ambiente Linux.

## 5.7 FreeBSD

O FreeBSD [23] é um sistema operacional *UNIX-like* gratuito, tido como robusto e seguro. Foi usado para testes do SCT, e também é o hospedeiro do OpenHSMd[24] utilizado no MRS[7].

## 5.8 C++

C++ é uma linguagem de programação orientada a objetos que se destaca por sua portabilidade, compatibilidade com código C e velocidade. Foi criada por Bjarne Stroustrup [8] enquanto trabalhava nos laboratórios Bell, mas recebeu este nome só no fim de 1983. Em outubro de 1985 foi lançada a primeira versão comercial da linguagem, assim como a primeira edição do manual “A linguagem de programação C++”, escrito por Stroustrup. O SCT foi implementado usando C++, assim como a biblioteca LibCryptoSec [10].

## 5.9 Java

Java[25] é tanto uma linguagem de programação de alto nível e orientada a objeto quanto uma plataforma. Em Java o código-fonte é compilado de texto plano para um código intermediário, o *bytecode*. Este código intermediário é executado em cima da plataforma Java, que por sua vez pode ser usada em uma grande variedade de sistemas operacionais e arquiteturas, desde computadores desktop até celulares. Assim, pode-se escrever uma aplicação uma vez, e rodá-la em qualquer sistema que tenha uma plataforma Java instalada (a Máquina Virtual Java - JVM).

## 5.10 MySQL

O MySQL [26] é um banco de dados de código aberto, muito utilizado em stacks LAMP (Linux, Apache, Mysql, PHP/Perl/Python). É também reconhecido por sua confiabilidade, performance e facilidade de uso.

# Capítulo 6

# Considerações Finais

Na parte do trabalho destinada à implementação do servidor de carimbo de tempo, foram atingidos os objetivos geral e específicos estipulados na introdução, menos um: não foi possível integrar o servidor de carimbo de tempo com o *HSM* com sucesso até o momento, por conta de incompatibilidade entre a versão do OpenSSL utilizada no desenvolvimento da biblioteca *LibCryptoSec* (uma versão ainda sob desenvolvimento, usada por conter as extensões necessárias para manipular carimbos de tempo) e a versão usada no sistema do *HSM* (versão estável, mas sem as extensões de carimbo de tempo). Excetuando-se este percalço, o restante do trabalho correu como esperado, e os objetivos específicos foram alcançados.

Quanto à segunda parte, a implementação da aplicação emissora de certificados otimizados e da aplicação de gerenciamento de provas novomodo correu sem problemas intransponíveis. Esta parte do trabalho também serviu bem o propósito de permitir avaliar e refinar o estudo das especificações da infra-estrutura de chaves públicas de certificados otimizados, melhorando a compreensão geral das possibilidades e limitações desta solução. Também foi comprovada a validade das especificações dos certificados otimizados e suas extensões.

Como trabalhos futuros, pode-se sugerir:

- a integração definitiva do servidor de carimbo de tempo com o HSM, pois este processo depende também da atualização da biblioteca LibCryptoSec;

- a expansão das aplicações de suporte à ICP de certificados otimizados, incluindo integração com HSMs e a automatização de geração de certificados para a ACCO;
- estudo da divulgação automatizada das provas de novomodo.

# Referências


[1] CUSTóDIO, R. F. et al. Temporal key release infrastructure. 2007.

[2] KOHNFELDER, L. M. *Towards a practical public-key cryptosystem*. [S.l.], 1978.

[3] DZUNG, D. et al. Security for industrial communication systems. *Proceedings of the IEEE 93 (6) (2005) 1152–1177*, 2005.

[4] VIGIL, M. A. G. et al. Optimized public key infrastructure - a pki to support efficient document's signatures. 2009.

[5] CUSTóDIO, R. F. et al. Optimized certificates - a new proposal for efficient electronic document signature validation. 2008.

[6] MICALI, S. Novomodo: Scalable certificate validation and simplified pki management. *Proceedings of the 1st Annual PKI Research Workshop, NIST, Gaithersburg MD, USA*, 2002.

[7] ROMANI, J. *Integração de Serviços de Relógio para Infra-estrutura de Chaves Públicas*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, 2008.

[8] A.ELLIS, M.; STROUSTRUP, B. C++: manual de referência comentado. In: *C++: manual de referência comentado*. [S.l.]: Rio de Janeiro (RJ): Campus c1993., 1993.

[9] GROUP, T. P. *PHP: Hypertext Preprocessor*. June 2007.

[10] LABORATóRIO de Segurança em Informação - LibCryptoSec. Maio 2009. Disponível em: <http://projetos.labsec.ufsc.br/libcryptosec>.

[11] SCHNEIER, B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. [S.l.]: John Wiley and Sons, 1996.

[12] HOUSLEY, R.; POLK, T. *Planning for PKI - Best practices guide for deploying public key infrastructures*. [S.l.]: John Wiley and Sons.

[13] NATIONAL Institute of Standards and Technology. June 2007. Disponível em: <www.nist.gov>.

[14] DIFFIE, W.; HELLMANN, M. E. New directions on cryptographic techniques. *Proceedings of the AFIPS National Computer Conference*, 1976.

[15] IETF. *RFC 3161*. Maio 2009. Disponível em: <http://www.ietf.org/rfc/rfc3161.txt>.

[16] BAYER, D.; HABER, S.; STORNETTA, W. S. *Improving the Efficiency and Reliability of Digital Time-Stamping*. [S.l.], 1993.

[17] THE Linux Home Page at Linux Online. June 2007. Disponível em: <www.linux.org>.

[18] THE Apache Software Foundation. Maio 2009. Disponível em: <http://ws.apache.org/axis2/>.

[19] LEGION of the Bouncy Castle. Maio 2009. Disponível em: <http://bouncycastle.org/java.html>.

[20] DUBUISSON, O. Asn.1 - communication between heterogeneous systems. In: ____. Elsevier-Morgan Kaufmann. Disponível em: <http://www.oss.com/asn1/dubuisson.html>.

[21] W3C. *Simple Object Access Protocol (SOAP) 1.1 - W3C Note*. May 2007. Disponível em: <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>.

[22] OPENSSL: The Open Source toolkit for SSL/TLS. June 2007. Disponível em: <www.openssl.org>.

[23] THE FreeBSD Project. Maio 2009. Disponível em: <http://www2.br.freebsd.org/>.

[24] ICP-EDU OpenHSMd. Maio 2009. Disponível em: <https://projetos.labsec.ufsc.br/openhsmd>.

[25] ABOUT the Java Technology. Maio 2009. Disponível em: <http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html>.

[26] MYSQL: The world's most popular open source database. Maio 2009. Disponível em: <http://www.mysql.com/>.

# Capítulo 7

# Anexo A: Códigos-Fonte

## 7.1 Servidor de Carimbo de Tempo

### 7.1.1 Arquivo: FileIO.h

```
 #ifndef FILEIO_H_
#define FILEIO_H_

#include <fstream>
#include <string>
#include "exception/FileSystemException.h"

class FileIO {
public:
  FileIO();
  virtual ~FileIO();

  static void readFile(std::string path, std::string& data)
      throw (FileSystemException) {
    ifstream file(path.c_str(), ios::in | ios::binary | ios::ate);
    char* memblock;
    ifstream::pos_type size;

    if (!file.is_open()) {
      throw FileSystemException(FileSystemException::OPEN,
          "FileIO::readFile");
    }

    size = file.tellg();
```

```
    memblock = new char[size];
    file.seekg(0, ios::beg);
    file.read(memblock, size);
    data = std::string(memblock, size);

    file.close();
    delete[] memblock;

  }

  static void writeFile(std::string path, char const * from,
      unsigned int size) throw (FileSystemException) {
    ofstream dest;

    dest.open(path.c_str(), ios::out | ios::trunc | ios::binary);

    if (dest.bad()) {
      throw FileSystemException(FileSystemException::OPEN,
          "Utils::readBinFile");
    }

    dest.write(from, size);

    dest.close();
  }

};

#endif /*FILEIO_H_*/
```

### 7.1.2 Arquivo: FileSystemException.h

```
 #ifndef FILESYSTEMEXCEPTION_H_
#define FILESYSTEMEXCEPTION_H_

#include <exception/LibCryptoSecException.h>

using namespace std;

class FileSystemException: public LibCryptoSecException {

public:
```

```
enum ErrorCode {
  UNKNOWN, FILENOTFOUND, CORRUPTEDFILE, OPEN, READ, WRITE
};

FileSystemException(string where) throw () {
  this->error = UNKNOWN;
  this->where = where;
}

FileSystemException(ErrorCode error, string where) throw () {
  this->error = error;
  this->where = where;
}

FileSystemException(string details, ErrorCode error, string where) throw () {
  this->details = details;
  this->error = error;
  this->where = where;
}

virtual ~FileSystemException() throw () {
}

virtual std::string getMessage() const {
  return (FileSystemException::errorCode2Message(this->error));
}

virtual string toString() const {
  std::string ret;
  if (this->error == FileSystemException::UNKNOWN) {
    ret = "FileSystemException. Called by: " + this->where + ".";
  } else {
    ret = "FileSystemException: "
        + FileSystemException::errorCode2Message(this->error)
        + ". Called by: " + this->where + ".";
  }
  return ret;
}

virtual FileSystemException::ErrorCode getErrorCode() {
  return this->error;
}

static std::string errorCode2Message(FileSystemException::ErrorCode error) {
```

```
    std::string ret;
    switch (error) {
    case FileSystemException::UNKNOWN:
      ret = "Unknown error";
      break;
    case FileSystemException::FILENOTFOUND:
      ret = "File not found";
      break;
    case FileSystemException::CORRUPTEDFILE:
      ret = "Corrupted file";
      break;
    case FileSystemException::OPEN:
      ret = "Error while opening file";
      break;
    case FileSystemException::READ:
      ret = "Error while reading file";
      break;
    case FileSystemException::WRITE:
      ret = "Error while writing on file";
      break;
    }
    return ret;
  }

protected:
  ErrorCode error;
};

#endif /*FILESYSTEMEXCEPTION_H_*/
```

### 7.1.3 Arquivo: IPCSemaphore.cpp

```
#include "IPCSemaphore.h"

IPCSemaphore::IPCSemaphore() {
  this->id_semaforo = this->binary_semaphore_allocation(IPC_PRIVATE, S_IRUSR
      | S_IWUSR);
  this->binary_semaphore_initialize(this->id_semaforo, (unsigned short) 1);
}

IPCSemaphore::~IPCSemaphore() {
  std::cout << "desalocando o semáforo!" << std::endl;
```

```
  this->binary_semaphore_deallocate(this->id_semaforo);
}

void IPCSemaphore::wait() {
  std::cout << "waiting" << std::endl;
  this->binary_semaphore_wait(this->id_semaforo);
}

void IPCSemaphore::signal() {
  std::cout << "signalling" << std::endl;
  this->binary_semaphore_post(this->id_semaforo);
}

/* Obtém um ID de um semáforo binário, alocando-o se necessário */
int IPCSemaphore::binary_semaphore_allocation(key_t key, int sem_flags) {
  return semget(key, 1, sem_flags);
}

/* Desaloca um semáforo binário. Todos os processos que o usam já
 * devem ter terminado de usá-lo. Retorna -1 em caso de falha */
int IPCSemaphore::binary_semaphore_deallocate(int semid) {
  union semun ignored_argument;
  return semctl(semid, 1, IPC_RMID, ignored_argument);
}

/* Inicializa um semáforo binário com valor passado no argumento <inicial> */
int IPCSemaphore::binary_semaphore_initialize(int semid, unsigned short inicial) {
  union semun argument;
  unsigned short values[1];
  //  values[0] = 1;
  std::cout << "inicializando semáforo com " << inicial << std::endl;
  values[0] = inicial;
  argument.array = values;
  return semctl(semid, 0, SETALL, argument);
}

/* Wait on a binary semaphore.  Block until the semaphore value is positive, then
 decrement it by 1. */
/* Wait num semáforo binário. Bloqueia até que o valor do semáforo seja positivo,
 * e então decrementa-o em 1. */
int IPCSemaphore::binary_semaphore_wait(int semid) {
  struct sembuf operations[1];
  /* Use the first (and only) semaphore.  */
  operations[0].sem_num = 0;
```

```
  /* Decrement by 1. */
  operations[0].sem_op = -1;
  /* Permit undo'ing. */
  operations[0].sem_flg = SEM_UNDO;
  return semop(semid, operations, 1);
}

/* Post to a binary semaphore: increment its value by 1.
 This returns immediately. */
/* Post (signal) num semáforo binário: incrementa seu valor em 1.
 * Retorna imediatamente.*/
int IPCSemaphore::binary_semaphore_post(int semid) {
  struct sembuf operations[1];
  /* Use the first (and only) semaphore.  */
  operations[0].sem_num = 0;
  /* Increment by 1. */
  operations[0].sem_op = 1;
  /* Permit undo'ing. */
  operations[0].sem_flg = SEM_UNDO;
  return semop(semid, operations, 1);
}
```

### 7.1.4 Arquivo: IPCSemaphore.h

```
#ifndef IPCSEMAPHORE_H_
#define IPCSEMAPHORE_H_

#ifdef __FreeBSD__
#include <sys/time.h>
#include <sys/resource.h>
#endif

#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <iostream>

#ifdef __linux__
/* No FreeBSD a union semun já está definida em sys/sem.h */
```

```
union semun {
  int val;
  struct semid_ds *buf;
  unsigned short int *array;
  struct seminfo *__buf;
};
#endif

class IPCSemaphore {
public:
  IPCSemaphore();
  virtual ~IPCSemaphore();

  void wait();
  void signal();

private:
  /* Obtain a binary semaphore's ID, allocating if necessary. */
  int binary_semaphore_allocation(key_t key, int sem_flags);

  /* Deallocate a binary semaphore. All users must have finished their
   use. Returns -1 on failure. */
  int binary_semaphore_deallocate(int semid);

  /* Initialize a binary semaphore with a value of 1. */
  /* Inicializa um semáforo binário com valor passado no argumento <inicial> */
  int binary_semaphore_initialize(int semid, unsigned short inicial);

  /* Wait on a binary semaphore.  Block until the semaphore value is positive, then
   decrement it by 1. */
  int binary_semaphore_wait(int semid);

  /* Post to a binary semaphore: increment its value by 1.
   This returns immediately. */
  int binary_semaphore_post(int semid);

  int id_semaforo;
};

#endif /* IPCSEMAPHORE_H_ */
```

### 7.1.5 Arquivo: TimeStampServer.cpp

```
#include "TimeStampServer.h"

bool finalizar = false;

void unexpected_handler_meu(int s) {
  cout << "terminando com unexpected!" << endl;
  finalizar = true;
}

void terminate_handler_meu(int s) {
  cout << "terminando com terminate!" << endl;
  finalizar = true;
}
void ctrlc_handler_meu(int s) {
  cout << "terminando com ctrl-c!" << endl;
  finalizar = true;
}

TimeStampServer::TimeStampServer(std::string chave, std::string certificado,
    unsigned short int porta) {

  signal(SIGTERM, terminate_handler_meu);
  signal(SIGQUIT, ctrlc_handler_meu);

  this->semaforo = new IPCSemaphore();
  portaEscuta = porta;
  /*Reads certificate from disk*/
  string certData;
  string keyData;
  try {
    cout << "Lê o certificado" << endl;
    FileIO::readFile(certificado, certData);
    /*Reads private key from a file on the disk*/
    cout << "Lê a chave" << endl;
    FileIO::readFile(chave, keyData);
  } catch (FileSystemException ex) {
    cout
        << "Error reading the file containing the key and/or certificate, aborting"
        << endl;
    cout << "Certificate file: " << certificado.c_str() << endl;
    cout << "Key file: " << chave.c_str() << endl;
    cout << "Reason: " << ex.what() << endl;
```

```
    exit(1);
  }
  try {
    cert = new Certificate(certData);
    privKey = new PrivateKey(keyData);
  } catch (EncodeException ex) {
    cout
        << "erro na criação do certificado e/ou chave, impossível continuar"
        << endl;
    cout << "Error creating certificate and/or key, aborting" << endl;
    cout << "Reason: " << ex.what() << endl;
    exit(1);
  }
}

TimeStampServer::TimeStampServer(std::string ip, std::string engine_path,
    std::string chave, std::string certificado, unsigned short int porta) {

  signal(SIGTERM, terminate_handler_meu);
  signal(SIGQUIT, ctrlc_handler_meu);

  this->semaforo = new IPCSemaphore();

  portaEscuta = porta;
  /*Reads certificate from disk*/
  string certData;
  cout << "Reads certificate from disk\n";

  FileIO::readFile(certificado, certData);
  cert = new Certificate(certData);

  cout << "Create Engine\n";
  Engines::loadDynamicEngineSupport();
  engine = new DynamicEngine(engine_path);

  /*
   * Steps for reading the private key from the HSM
   * - engine constructor, using the engine in /usr/lib
   * - set the engine's parameters to connect to the HSM
   * (ADDRESS_CONN= HSM's ip e PORT_CONN= HSM's port to connect to)
   * - Key constructor, using the key ID and the engine:
   * -- KeyPair::KeyPair(Engine *engine, std::string keyId)
   * - method that returns the private key
   * -- PrivateKey* KeyPair::getPrivateKey()
```

```
   * */

  cout << "Sending commands to HSM.\n";
  string key;
  key = "ADDRESS_CONN";
  engine->setCommand(key, ip);
  //  key="PORT_CONN";
  //  engine->setCommand(key,porta_hsm);
  if (engine->testInit()) {
    cout << "Reading private key.\n";
    cout << "Trying to fetch key " << chave << " with engine "
        << engine_path << "\n";
    KeyPair* parDeChaves = new KeyPair(engine_path, chave);
    cout << "Getting private key...\n";
    privKey = parDeChaves->getPrivateKey();
  } else
    cout
        << "Error initializing the program, impossible to"
            << " fetch private key from HSM";
}

TimeStampServer::~TimeStampServer() {
  delete semaforo;
  close(listenSocket);
  delete cert;
  delete privKey;
  if (engine)
    delete engine;
  exit(0);
}

void sigchld_handler(int s) {
  cout << "child terminando" << endl;
  while (waitpid(-1, NULL, WNOHANG) > 0)
    ;
}

TimeStampResponse * TimeStampServer::geraResposta(ByteArray * requisicao) {

  //requisition from POST
  cout << "creating requisition" << endl;
  TimeStampRequest * ts_requisicao = new TimeStampRequest(*requisicao);
  cout << "creating response" << endl;
  TimeStampResponseBuilder builder1(*ts_requisicao);
```

```
  cout << "response builder created" << endl;

  ObjectIdentifier obj = ObjectIdentifierFactory::getObjectIdentifier(
      "1.3.6.1.4.1.14975.2.1.0");
  BigInteger serial("15");
  TimeStampResponse::Accuracy acc;
  vector<const Certificate*> trusted;

  acc.seconds = 100;
  acc.millis = 200;
  acc.micros = 300;

  builder1.setSigner(*cert, *privKey);

  builder1.setSerial(serial);
  builder1.setTSA(true);
  builder1.setOrdering(true);
  builder1.setAccuracy(acc);

  builder1.setDefaultPolicy(obj);

  return builder1.doFinal();
}

int TimeStampServer::determinaContentLength(string *requisicao) {
  int retorno = 0;
  unsigned int inicioLength = requisicao->find("Content-Length: ", 0) + 16;
  unsigned int fimLength = 0;
  if (inicioLength != string::npos) {
    fimLength = requisicao->find("\r\n", inicioLength);
    if (fimLength != string::npos) {
      retorno = atoi(requisicao->substr(inicioLength, fimLength
          - inicioLength).c_str());
    } else {
      cout << "error finding ending" << endl;
    }
  } else {
    cout << "error finding start" << endl;
  }
  return retorno;
}
int TimeStampServer::determinaTamanhoHeader(string *requisicao) {
  //All lines in the header end with \r\n, and the header itself ends in a
  //newline containing only \r\n
```

```
  string fimHeader = "\r\n\r\n";
  return requisicao->find(fimHeader, 0) + fimHeader.size();
}

/**
 * Generates the response header to be sent back to the client.
 * int tamanhoConteudo - the size of the response
 * Returns a string containing the response
 * */
std::string TimeStampServer::geraHeaderResposta(int tamanhoConteudo) {
  std::ostringstream respHTTP;
  char hora[80];
  time_t rawtime;
  struct tm * timeinfo;
  time(&rawtime);
  timeinfo = gmtime(&rawtime);
  strftime(hora, 80, "%a, %d %b %Y %H:%M:%S %Z", timeinfo);
  string hora2;
  hora2.assign(hora);
  respHTTP << "HTTP/1.1 200 OK\r\n";
  respHTTP << "Server: Makeshift TS_Server\r\n";
  respHTTP << "Keep-Alive: timeout=5, max=100\r\n";
  respHTTP << "Date: " << hora2 << "\r\n";
  respHTTP << "Connection: Keep-Alive\r\n";
  respHTTP << "Character-Encoding: binary\r\n";
  respHTTP << "Content-Type: application/timestamp-reply\r\n";
  respHTTP << "Content-Length: " << tamanhoConteudo << "\r\n";
  //.tsr file size, mandatory
  respHTTP << "\r\n";//last line, contains only \r\n //mandatory
  return respHTTP.str();
}

/**
 * Gets the server online and in a listening state. Runs until finalizar is true.
 * */
void TimeStampServer::run() {

  int connectSocket;

  socklen_t clientAddressLength;
  struct sockaddr_in clientAddress, serverAddress;
  char line[LINE_ARRAY_SIZE];

  listenSocket = socket(AF_INET, SOCK_STREAM, 0);
```

```
if (listenSocket < 0) {
  cerr << "Erro na criação do listenSocket" << endl;
  exit(1);
}

serverAddress.sin_family = AF_INET;
serverAddress.sin_addr.s_addr = htonl(INADDR_ANY);
serverAddress.sin_port = htons(portaEscuta);

int yes = 1;
// lose the pesky "Address already in use" error message
if (setsockopt(listenSocket, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int))
    == -1) {
  perror("setsockopt");
  exit(1);
}

if (bind(listenSocket, (struct sockaddr *) &serverAddress,
    sizeof(serverAddress)) < 0) {
  cerr << "cannot bind socket";
  exit(1);
}

//keeps 5 connections in the queue
listen(listenSocket, 5);

sa.sa_handler = sigchld_handler; // reap all dead processes
sigemptyset(&sa.sa_mask);
sa.sa_flags = SA_RESTART;
if (sigaction(SIGCHLD, &sa, NULL) == -1) {
  perror("sigaction");
  exit(1);
}

while (1 && !finalizar) {

  cout << "Waiting for connections at " << portaEscuta << " ...\n";
  clientAddressLength = sizeof(clientAddress);

  connectSocket = accept(listenSocket,
      (struct sockaddr *) &clientAddress, &clientAddressLength);
  if (connectSocket < 0) {
    cerr << "Error accepting connection.";
    exit(1);
```

```
}
if (!fork()) { // this is the child process
  semaforo->wait();
  close(listenSocket); // child doesn't need the listener
  cout << "  connected to " << inet_ntoa(clientAddress.sin_addr);
  cout << ":" << ntohs(clientAddress.sin_port) << "\n";
  memset(line, 0x0, LINE_ARRAY_SIZE);
  //this part works the magic
  while (recv(connectSocket, line, MAX_MSG, 0) > 0) {
    string req_completa;
    req_completa.assign(line);

    int tamanhoHeader = determinaTamanhoHeader(&req_completa);
    int contentLength = determinaContentLength(&req_completa);
    cout << "header size:: " << tamanhoHeader
        << " , contentLength: " << contentLength << "\n";

    ByteArray * req = new ByteArray(
        (const unsigned char *) &line[tamanhoHeader],
        (unsigned int) contentLength);
    cout << "req. created\n";
    TimeStampResponse * resp = geraResposta(req);
    string cabecalho = geraHeaderResposta(
        resp->getDerEncoded().size());

    int tamanhoResposta = resp->getDerEncoded().size()
        + cabecalho.size();
    char buf[tamanhoResposta];
    memcpy(buf, cabecalho.c_str(), cabecalho.size());
    memcpy((char *) &buf[cabecalho.size()],
        resp->getDerEncoded().getDataPointer(),
        resp->getDerEncoded().size());

    // Send response back to client
    if (send(connectSocket, buf, tamanhoResposta, 0) < 0)
      cerr << "Error: cannot send modified data";

    memset(line, 0x0, LINE_ARRAY_SIZE); // set line to all zeroes
    close(connectSocket);
  }
  semaforo->signal();
  exit(0);
}
close(connectSocket); // parent doesn't need this
```

```
  }
  //exits normally
  close(listenSocket);
  exit(0);
}
```

### 7.1.6 Arquivo: TimeStampServer.h

```
#ifndef TIMESTAMPSERVER_H_
#define TIMESTAMPSERVER_H_

#ifdef __FreeBSD__
#include <sys/socket.h>
#endif

#include <iostream>
#include <vector>
#include <arpa/inet.h>
#include <netdb.h>
#include <netinet/in.h>
#include <unistd.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <sys/wait.h>
#include <csignal>

#include "ts/TimeStampResponseBuilder.h"
#include "lib/FileIO.h"
#include "lib/FileSystemException.h"
#include "MessageDigest.h"
#include <ctime>
#include <stdio.h>
#include <time.h>

#include "Engine.h"
#include "Engines.h"
#include "DynamicEngine.h"
#include "KeyPair.h"

#include "ByteArray.h"
#include "BigInteger.h"
```

```
#include "certificate/ObjectIdentifierFactory.h"
#include "exception/CertificationException.h"
#include "exception/EncodeException.h"

#include "certificate/Extension.h"
#include "certificate/KeyUsageExtension.h"
#include "certificate/ExtendedKeyUsageExtension.h"
#include "certificate/BasicConstraintsExtension.h"
#include "certificate/CRLDistributionPointsExtension.h"
#include "certificate/IssuerAlternativeNameExtension.h"
#include "certificate/SubjectAlternativeNameExtension.h"
#include "certificate/AuthorityKeyIdentifierExtension.h"
#include "certificate/SubjectKeyIdentifierExtension.h"
#include "certificate/CertificatePoliciesExtension.h"
#include "certificate/CRLNumberExtension.h"

#include <openssl/asn1.h>
#include <openssl/asn1t.h>
#include <openssl/stack.h>
#include <openssl/x509.h>
#include <openssl/x509v3.h>
#include <openssl/objects.h>
#include <openssl/ts.h>

#include "lib/IPCSemaphore.h"

using namespace std;

#define MAX_MSG 400
#define LINE_ARRAY_SIZE (MAX_MSG+1)

class TimeStampServer {
public:

  /**
   * Constructor which gets the private key from an external HSM
   * @param ip The HSMs ip address
   * @param engine The path to the engine file
   * (e.g.: /usr/lib/engine_openhsmd.so)
   * @param chave The ID of the private key
   * @param certificado Path to the certificate file on the disk
   * @param porta Port on which the server will listen

   * Construtor usando HSM externo para pegar a chave
```

```
     * @param ip O endereço IP do HSM
     * @param engine O caminho para a engine no sistema
     * (ex.: /usr/lib/engine_openhsmd.so)
     * @param chave O id da chave a ser usada
     * @param certificado O caminho do certificado no disco
     * @param porta A porta em que o servidor ficará ouvindo requisições
     * */
    TimeStampServer(std::string ip, std::string engine, std::string chave,
        std::string certificado, unsigned short int porta);

    /**
     * Constructor using certificate and key files found on the disk (without HSM)
     * @param chave Path to the private key file
     * @param certificado Path to the certificate file
     * @param porta Port on which the server will listen

     * Construtor usando chave e certificado armazenados no disco
     * @param chave O caminho da chave privada no disco
     * @param certificado O caminho do certificado no disco
     * @param porta A porta em que o servidor ficará ouvindo requisições
     * */
    TimeStampServer(std::string chave, std::string certificado,
        unsigned short int porta);

    virtual ~TimeStampServer();

    /*handles the zombies (suggested parameters: shotgun,
    axe, flamethrower*/
    //  void sigchld_handler(int s);

    /*
     * @brief Runs the server
     * */
    void run();

  private:
    std::string geraHeaderResposta(int tamanhoConteudo);

    /* Retorna o tamanho do cabeçalho HTTP da requisição recebida
     * @param a string contendo o cabeçalho

     /* Returns the size of the header
     * @param the string containing the request
     * */
```

```
  int determinaTamanhoHeader(string *requisicao);

  /* Gets the content-length field from the header
   * @param the string containing the request

   /* Retorna o campo content-length do cabeçalho da requisição
   * @param requisicao String contendo a requisição
   * */
  int determinaContentLength(string *requisicao);

  TimeStampResponse * geraResposta(ByteArray * requisicao);
  string ip_hsm, porta_hsm, engine_path, id_chave;
  Engine* engine;
  PrivateKey* privKey;
  Certificate* cert;
  unsigned short int portaEscuta;
  int listenSocket;
  struct sigaction sa;
  IPCSemaphore* semaforo;
};

#endif /* TIMESTAMPSERVER_H_ */
```

### 7.1.7 Arquivo: ConfiguracaoArquivo.cpp

```
#include "ConfiguracaoArquivo.h"
#include "../FileSystemException.h"

using namespace std;

/*
 * Reads a configuration file with entries in the
 * identifier=value format, each on a line
 * @param fonte absolute path to the configuration file

 * Lê um arquivo de configuração com entradas no formato
 * identificador=valor, separados por quebra de linha (padrão Unix, \n)
 * @param fonte caminho para o arquivo de configuração (absoluto)
 * */

ConfiguracaoArquivo::ConfiguracaoArquivo(string fonte) {
  cout << fonte << endl;
```

```
}

bool ConfiguracaoArquivo::lerConfiguracao(string fonte) {
  bool retorno = false;
  string dados = "";
  try {
    FileIO::readFile(fonte, dados);
  } catch (FileSystemException ex) {
    cout << "Unable to open the config file!" << endl;
    exit(1);
  }
  cout << "Data read from file " << fonte << endl;
  int posAtual = 0;
  do {
    string novaLinha = leiaLinha(&dados, &posAtual);
    if (novaLinha != "" && novaLinha != "\n")
      adicioneEntrada(novaLinha);
  } while (posAtual > 0);
  return retorno;
}

bool ConfiguracaoArquivo::salvarConfiguracao(string destino) {
  bool retorno = false;
  map<string, string>::iterator it;
  stringbuf confs;

  for (it = configs.begin(); it != configs.end(); it++) {
    string temp = (*it).first;
    temp.append("=");
    temp.append((*it).second);
    temp.append("\n");
    confs.sputn(temp.c_str(), temp.size());
  }
  try {
    FileIO::writeFile(destino, ((string) confs.str()).c_str(),
        ((string) confs.str()).size());
  } catch (FileSystemException ex) {
    cout << "Error writing new config file" << endl;
    cout << ex.what() << endl;
  }
  return retorno;
}

/*Reads a line*/
```

```
string ConfiguracaoArquivo::leiaLinha(string *dados, int *posAtual) {
  string retorno = "";
  if (*posAtual < int(dados->size())) {
    int proxPos = dados->find("\n", size_t(*posAtual));
    if (size_t(proxPos) != string::npos) {
      retorno = dados->substr(size_t(*posAtual), size_t(proxPos
          - (*posAtual)));
    } else {
      proxPos = dados->rfind("\0");
      if (proxPos > *posAtual) {
        retorno = dados->substr(size_t(*posAtual), size_t(proxPos
            - (*posAtual)));
      } else {
        *posAtual = -1;
      }
    }
    *posAtual = proxPos + 1;
  } else {
    *posAtual = -1;
  }

  return retorno;
}

void ConfiguracaoArquivo::adicioneEntrada(string linha) {
  unsigned int posIgual = linha.find("=");
  map<string, string>::iterator it;
  it = configs.end();
  if (posIgual != string::npos) {
    string chave = linha.substr(0, posIgual);
    string valor = linha.substr(posIgual + 1, linha.length());
    configs.insert(it, pair<string, string> (chave, valor));
  }
}

string ConfiguracaoArquivo::retorneValorDe(string nome) {
  string retorno;
  map<string, string>::iterator it;
  it = configs.find(nome);
  if (it != configs.end()) {//item foi encontrado
    retorno = (*it).second;
  } else {
    retorno = "";
  }
```

```
  return retorno;
}
```

### 7.1.8 Arquivo: ConfiguracaoArquivo.h

```
#ifndef CONFIGURACAOARQUIVO_H_
#define CONFIGURACAOARQUIVO_H_

#include "Configuracao.h"
#include "../FileIO.h"

using namespace std;

class ConfiguracaoArquivo: public Configuracao {
public:

  ConfiguracaoArquivo();
  ConfiguracaoArquivo(string dummy);
  ~ConfiguracaoArquivo();

  /*
   * Reads a configuration file with entries in the
   * identifier=value format, each on a line
   * @param fonte absolute path to the configuration file

   * Lê um arquivo de configuração com entradas no formato
   * identificador=valor, separados por quebra de linha (padrão Unix, \n)
   * @param fonte caminho para o arquivo de configuração (absoluto)
   * */
  bool lerConfiguracao(string fonte);

  bool salvarConfiguracao(string destino);

  /* Searches for a configuration and returns its value */
  /* Procura um nome nas configurações carregadas e retorna o valor associado */
  string retorneValorDe(string nome);

private:
  string leiaLinha(string *dados, int *posAtual);

  void adicioneEntrada(string linha);
};
```

```
#endif /* CONFIGURACAOARQUIVO_H_ */
```

### 7.1.9 Arquivo: Configuracao.h

```
#ifndef CONFIGURACAO_H_
#define CONFIGURACAO_H_

#define ENTRADA_NAO_ENCONTRADA  "-1";

#include <iostream>
#include <map>

using namespace std;

class Configuracao {
public:

  /* Reads the configuration from the specified file */
  /* Lê as configuração a partir da fonte indicada */
  virtual bool lerConfiguracao(string fonte)=0;

  /* Saves the configuration to a file specified by destino */
  /* Salva as configurações carregadas para o destino indicado */
  virtual bool salvarConfiguracao(string destino)=0;

  /* Gets the value for nome from the configs */
  /* Procura o valor correspondente a nome nas configurações carregadas */
  virtual string retorneValorDe(string nome)=0;

protected:

  map<string, string> configs; //map containing the configs
};

#endif /* CONFIGURACAO_H_ */
```

### 7.1.10 Arquivo: main.cc

```
#include <pthread.h>
#include <stdio.h>
#include <getopt.h>
#include <map>

#include "TimeStampServer.h"
#include "lib/IPCSemaphore.h"
#include "lib/FileIO.h"
#include "lib/conf/ConfiguracaoArquivo.h"

//needed for daemonizing

#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <syslog.h>
#include <errno.h>
#include <pwd.h>
#include <signal.h>

/* Change this to the user under which to run */
string run_as;

#define EXIT_SUCCESS 0
#define EXIT_FAILURE 1

string chave_path, cert_path, conf_path, nome, nome_lock_file;
//chave_path="chavePrivada_ts.pem";
//cert_path="cert_TS.cer";
unsigned short int porta;

static void child_handler(int signum) {
  switch (signum) {
  case SIGALRM:
    exit(EXIT_FAILURE);
    break;
  case SIGUSR1:
    exit(EXIT_SUCCESS);
    break;
  case SIGCHLD:
    exit(EXIT_FAILURE);
```

```
    break;
  }
}

static void daemonize(const char *lockfile) {
  cout << "starting daemonize" << endl;
  pid_t pid, sid, parent;
  int lfp = -1;

  /* already a daemon */
  if (getppid() == 1)
    return;
  syslog(LOG_INFO, "is already a daemon, return");
  /* Create the lock file as the current user */
  if (lockfile && lockfile[0]) {
    lfp = open(lockfile, O_RDWR | O_CREAT, 0640);
    if (lfp < 0) {
      syslog(LOG_ERR, "unable to create lock file %s, code=%d (%s)",
          lockfile, errno, strerror(errno));
      exit(EXIT_FAILURE);
    }
  }
  syslog(LOG_INFO, "changing user");
  /* Drop user if there is one, and we were run as root */
  if (getuid() == 0 || geteuid() == 0) {
    struct passwd *pw = getpwnam(run_as.c_str());
    if (pw) {
      string temp = "changing user to ";
      temp.append(run_as);
      syslog(LOG_INFO, temp.c_str());
      setuid(pw->pw_uid);
    }
  }

  /* Trap signals that we expect to recieve */
  signal(SIGCHLD, child_handler);
  signal(SIGUSR1, child_handler);
  signal(SIGALRM, child_handler);

  /* Fork off the parent process */
  pid = fork();
  if (pid < 0) {
    syslog(LOG_ERR, "unable to fork daemon, code=%d (%s)", errno, strerror(
        errno));
```

```
  exit(EXIT_FAILURE);
}
/* If we got a good PID, then we can exit the parent process. */
if (pid > 0) {
  syslog(LOG_INFO, "forked successfully (parent)");
  /* Wait for confirmation from the child via SIGTERM or SIGCHLD, or
   for two seconds to elapse (SIGALRM).  pause() should not return. */
  alarm(2);
  pause();

  exit(EXIT_FAILURE);
}

/* At this point we are executing as the child process */
parent = getppid();
syslog(LOG_INFO, "forked successfully (child)");
/* Cancel certain signals */
signal(SIGCHLD, SIG_DFL); /* A child process dies */
signal(SIGTSTP, SIG_IGN); /* Various TTY signals */
signal(SIGTTOU, SIG_IGN);
signal(SIGTTIN, SIG_IGN);
signal(SIGHUP, SIG_IGN); /* Ignore hangup signal */
signal(SIGTERM, SIG_DFL); /* Die on SIGTERM */

/* Change the file mode mask */
umask(0);

/* Create a new SID for the child process */
sid = setsid();
if (sid < 0) {
  syslog(LOG_ERR, "unable to create a new session, code %d (%s)", errno,
      strerror(errno));
  exit(EXIT_FAILURE);
}
syslog(LOG_INFO, "changing dir");
/* Change the current working directory.  This prevents the current
 directory from being locked; hence not being able to remove it. */
//    if ((chdir("/")) < 0) {
if ((chdir("/home/kas/dev/workspace-cpp/ts_server2/Debug")) < 0) {
  syslog(LOG_ERR, "unable to change directory to %s, code %d (%s)", "/",
      errno, strerror(errno));
  exit(EXIT_FAILURE);
}
syslog(LOG_INFO, "muting");
```

```
  /* Redirect standard files to /dev/null */
  freopen("/dev/null", "r", stdin);
  freopen("/dev/null", "w", stdout);
  freopen("/dev/null", "w", stderr);

  syslog(LOG_INFO, "Killing parent process");
  /* Tell the parent process that we are A-okay */
  kill(parent, SIGUSR1);
}

void* run_ts_server(void*) {
  //syslog( LOG_INFO, "inicializando método da thread" );
  syslog(LOG_INFO, "Initializing TimeStampServer");
  TimeStampServer* ts = new TimeStampServer(chave_path, cert_path, porta);
  syslog(LOG_INFO, "TimeStampServer Initialized");
  ts->run();
  delete ts;
  cout << "exiting" << endl;
  return (void*) 0;
}

void help() {
  cout << "Usage: ts_server -c <config-file>" << endl;
  cout << "Configuration File: entries are expected "
      << "to be in the name=value format" << endl;
}

bool inicializa_parametros(Configuracao* conf) {
  bool retorno = true;
  string temp = "reading configs from file: ";
  temp.append(conf_path.c_str());
  syslog(LOG_INFO, temp.c_str());
  string cert_path_nome = "cert_path";
  string chave_path_nome = "chave_path";
  string porta_nome = "porta";
  string nome_daemon = "nome";
  string rodar_como = "rodar_como";
  string lock_file = "lock_file";

  conf->lerConfiguracao(conf_path);
  //  if((cert_path=conf->retorneValorDe("cert_path"))==ENTRADA_NAO_ENCONTRADA){
  //    retorno=false;
  //  }
```

```
  nome = conf->retorneValorDe(nome_daemon);
  cout << "nome: " << nome << endl;
  run_as = conf->retorneValorDe(rodar_como);
  cout << "run_as: " << run_as << endl;
  nome_lock_file = conf->retorneValorDe(lock_file);
  cout << "nome_lock_file: " << nome_lock_file << endl;
  cert_path = conf->retorneValorDe(cert_path_nome);
  cout << "cert_path: " << cert_path << endl;
  chave_path = conf->retorneValorDe(chave_path_nome);
  cout << "chave_path: " << chave_path << endl;
  porta = atoi(conf->retorneValorDe(porta_nome).c_str());
  cout << "porta: " << porta << endl;

  return retorno;
}

int main(int argc, char **argv) {

  if (argc < 1) {
    help();
    exit(1);
  }
  static struct option long_options[] = { { "debug", no_argument, 0, '0' }, {
      "conf", required_argument, 0, '1' } };

  bool debug = false;
  int c = 0, option_index = 0;
  ;
  /* Leitura de parâmetros e leitura de configurações */
  while ((c = getopt_long_only(argc, argv, "", long_options, &option_index))
      != -1) {
    switch (c) {
    case '0':
      debug = true;
      break;
    case '1':
      conf_path = optarg;
      break;
    }
  }

  if (!debug) {
    /* Initialize the logging interface */
    openlog(nome.c_str(), LOG_PID, LOG_LOCAL5);
```

```
    syslog(LOG_INFO, "starting");
    /* Daemonize */
    daemonize(nome_lock_file.c_str());
    closelog();
  }
  openlog(nome.c_str(), LOG_PID, LOG_LOCAL5);
  syslog(LOG_INFO, "daemonizing finished, starting");

  string dummy = "teste";
  ConfiguracaoArquivo* conf = new ConfiguracaoArquivo(dummy);
  if (inicializa_parametros(conf)) { //inicializou os parametros com sucesso
    pthread_t thread_id;
    syslog(LOG_INFO, "prestes a criar thread");
    pthread_create(&thread_id, NULL, &run_ts_server, NULL);
    syslog(LOG_INFO, "criou thread, vai dar join");
    pthread_join(thread_id, NULL);
    syslog(LOG_INFO, "deu join na thread");
  } else {
    syslog(LOG_ERR, "error loading configs, aborting");
    exit(1);
  }
  /* Finish up */
  syslog(LOG_INFO, "terminated");
  closelog();
  return 0;

}
```

## 7.2 Certificados Otimizados

### 7.2.1 Classe: ClienteWS

```
package ws.jocca.normal;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.security.InvalidParameterException;
```

```
import java.security.KeyFactory;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.Signature;
import java.security.cert.CRLException;
import java.security.cert.CertificateEncodingException;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.cert.X509CRL;
import java.security.cert.X509Certificate;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.PKCS8EncodedKeySpec;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Scanner;
import java.util.logging.Logger;

import org.apache.axiom.om.OMAbstractFactory;
import org.apache.axiom.om.OMElement;
import org.apache.axiom.om.OMNamespace;
import org.apache.axiom.soap.SOAP11Constants;
import org.apache.axiom.soap.SOAPFactory;
import org.apache.axis2.AxisFault;
import org.apache.axis2.addressing.EndpointReference;
import org.apache.axis2.client.Options;
import org.apache.axis2.client.ServiceClient;
import org.bouncycastle.asn1.ASN1OctetString;
import org.bouncycastle.asn1.cms.Attribute;
import org.bouncycastle.asn1.cms.AttributeTable;
import org.bouncycastle.asn1.cms.CMSAttributes;
import org.bouncycastle.cms.CMSSignedData;
import org.bouncycastle.cms.SignerInformation;
import org.bouncycastle.cms.SignerInformationStore;
import org.bouncycastle.tsp.TimeStampToken;
import org.bouncycastle.util.encoders.Base64;

import ws.jocca.normal.lib.Util;

public class ClienteWS {

  /** Web Service URL */
  private static EndpointReference targetEPR;
```

```
/** Namespace (class' package "path", backwards) */
private static String nameSpace = "http://normal.jocca.ws";
static Logger log = Logger.getLogger(ClienteWS.class.getName());

X509Certificate certUser = null;
List<X509Certificate> certsIntermediarios;
List<X509CRL> crls;
List<TimeStampToken> carimbosDeTempo;
PrivateKey privkey = null;

/**
 * Inicializa o cliente usando os certificados e CRLs fornecidos pelo
 * usuário Initializes the client using the certificates, CRLs and other
 * needed parameters passed separately by the user.
 */
public ClienteWS(X509Certificate certUser,
    List<X509Certificate> certsIntermediarios, Signature sig,
    List<X509CRL> crls, List<TimeStampToken> ts, PrivateKey pkey) {

  try {
    if (leConfiguracaoDoDisco())
      this.inicializa(certUser, certsIntermediarios, crls, ts);
    else {// something necessary couldn't be found/read, quit
      log.severe("Not all configuration was done, aborting.");
    }
  } catch (FileNotFoundException e) {
    log.severe("Unable to open configuration file, aborting.");
    e.printStackTrace();
  }
  log.info("Class initialized successfully");
}

public ClienteWS() {
  try {
    if (!leConfiguracaoDoDisco()) {
      log.severe("Not all configuration was done, aborting.");
      System.exit(1);
    }
  } catch (FileNotFoundException e) {
    log.severe("Unable to open configuration file, aborting.");
    e.printStackTrace();
  }
  log.info("Class initialized successfully");
}
```

```
/**
 * Initializes the class (if you are not going to use a CMS)
 *
 * @param certUser
 *            User's certificate.
 * @param certsIntermediarios
 *            List of certificates which comprise the user's certificate's
 *            certification path.
 * @param crls
 *            List of CRLs used to validate the user's certificate.
 * @param ts
 *            List of Timestamp tokens
 * @throws InvalidParameterException
 * */

protected void inicializa(X509Certificate certUser,
    List<X509Certificate> certsIntermediarios, List<X509CRL> crls,
    List<TimeStampToken> ts) {

  if ((certUser != null) & (certUser instanceof X509Certificate)) {
    this.certUser = certUser;
  } else {
    throw new InvalidParameterException(
        "Invalid parameter: User's certificate is null or not a valid X509Certificate!");
  }
  if (certsIntermediarios == null || certsIntermediarios.size() < 1) {
    this.certsIntermediarios = new ArrayList<X509Certificate>();
  } else {
    this.certsIntermediarios = certsIntermediarios;
  }
  if (crls == null || crls.size() < 1) {
    this.crls = new ArrayList<X509CRL>();
  } else {
    this.crls = crls;
  }
  if (ts == null || ts.size() < 1) {
    this.carimbosDeTempo = new ArrayList<TimeStampToken>();
  } else {
    this.carimbosDeTempo = ts;
  }
}

/**
```

```
 * Gets the OCCA's certificate from the web service.
 *
 * @return the OCCA's certificate
 * */
public X509Certificate iniciaClienteObtemCertACCO() throws Exception {

  Options options = new Options();
  options.setTo(targetEPR);
  options.setAction("urn:retornaCertificadoACCO");
  options.setSoapVersionURI(SOAP11Constants.SOAP_ENVELOPE_NAMESPACE_URI);
  X509Certificate accoCert = null;
  try {
    ServiceClient sender = new ServiceClient();
    sender.setOptions(options);
    ByteArrayInputStream bais;
    CertificateFactory cf = CertificateFactory.getInstance("X.509");

    OMElement req = geraPayloadRetornaCertACCO();

    log.info("sender.sendReceive(geraCertOtimizado);");
    // blocking call to web service
    OMElement res = sender.sendReceive(req);

    // process the response
    try {
      bais = new ByteArrayInputStream(Base64.decode(res
          .getFirstElement().getText()));
      log.info("Gerando certificado retornado pelo serviço");
      accoCert = (X509Certificate) cf.generateCertificate(bais);
      log.info("Certificado gerado com sucesso!");
      // prints the received certificate
      log.info("Imprimindo certificado retornado");
      log.info(accoCert.toString());
      // write the certificate to the disk
      /*
       * FileOutputStream out = new FileOutputStream("cert-occa.pem");
       * out.write(accoCert.getEncoded()); out.flush();out.close();
       */
    } catch (CertificateException cex) {
      // an error occurred, print error message
      log
          .severe("Serviço retornou mensagem de erro. \nMensagem retornada: "
              + new String(Base64.decode(res
                  .getFirstElement().getText())));
```

```
    }
    log.info("retornou da chamada");
  } catch (AxisFault af) {
    af.printStackTrace();
  }
  return accoCert;
}

/**
 * Invokes the web service, using a CMS to pass the needed information.
 * */
public X509Certificate iniciaClienteCMS(File arquivoCMS) throws Exception {

  Options options = new Options();
  options.setTo(targetEPR);
  options.setAction("urn:geraCertOtimizadoCMS");
  options.setSoapVersionURI(SOAP11Constants.SOAP_ENVELOPE_NAMESPACE_URI);
  X509Certificate ocCert = null;
  try {
    ServiceClient sender = new ServiceClient();
    sender.setOptions(options);
    ByteArrayInputStream bais;
    CertificateFactory cf = CertificateFactory.getInstance("X.509");

    // reads the CMS from the disk
    FileInputStream fis = null;
    if (arquivoCMS != null)
      fis = new FileInputStream(arquivoCMS);
    // if no File is given, read test CMS
    else
      fis = new FileInputStream("cms-imagem-valido.pem");
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    int b = fis.read();

    while (b != -1) {
      baos.write(b);
      b = fis.read();
    }
    // Instantiates the CMS
    CMSSignedData cms = new CMSSignedData(baos.toByteArray());

    // jugglery to extract the signed document's hash from the CMS
    SignerInformationStore signers1 = cms.getSignerInfos();
    Iterator<SignerInformation> it1 = signers1.getSigners().iterator();
```

```
    SignerInformation signer1 = it1.next();
    AttributeTable table = signer1.getSignedAttributes();
    Attribute hashAttr = table.get(CMSAttributes.messageDigest);
    byte[] hash2 = ((ASN1OctetString) hashAttr.getAttrValues()
        .getObjectAt(0)).getOctets();
    // prints the hash
    // System.out.println("Hash: "+new String(Base64.encode(hash2)));

    OMElement geraCertOtimizadoCMS = geraPayloadCMS(cms.getEncoded(),
        hash2);

    log.info("sender.sendReceive(geraCertOtimizado);");
    // blocking call to web service
    OMElement res = sender.sendReceive(geraCertOtimizadoCMS);

    // process the response
    try {
      bais = new ByteArrayInputStream(Base64.decode(res
          .getFirstElement().getText()));
      log.info("Gerando certificado retornado pelo serviço");
      ocCert = (X509Certificate) cf.generateCertificate(bais);
      log.info("Certificado gerado com sucesso!");
      log.info("Imprimindo certificado retornado");
      log.info(ocCert.toString());
      // writes the optimized certificate to the disk
      /*
       * FileOutputStream out = new FileOutputStream("cert-oc.pem");
       * out.write(ocCert.getEncoded()); out.flush();out.close();
       */
    } catch (CertificateException cex) {
      // an error occurred, print the error message.
      log
          .severe("Service returned an error message. \nMessage returned: "
              + new String(Base64.decode(res
                  .getFirstElement().getText())));
    }
    log.info("returned from call");
  } catch (AxisFault af) {
    af.printStackTrace();
  }
  return ocCert;
}

/**
```

```
 * Invokes the web service passing the parameters separately. The class
 * needs to be initialized using the constructor
 * {@link ClienteWS#ClienteWS(X509Certificate, List, Signature, List, List, PrivateKey)}
 * or the {@link ClienteWS#inicializa(X509Certificate, List, List, List)}.
 *
 * @return The optimized certificate
 * */
public X509Certificate iniciaClienteSeparado() throws Exception {
  X509Certificate certUser, ocCert = null;

  Options options = new Options();
  options.setTo(targetEPR);
  options.setAction("urn:geraCertOtimizado");
  options.setSoapVersionURI(SOAP11Constants.SOAP_ENVELOPE_NAMESPACE_URI);

  try {
    ServiceClient sender = new ServiceClient();
    sender.setOptions(options);

    OMElement geraCertOtimizado = geraPayload(this.certUser,
        this.certsIntermediarios, this.crls, this.carimbosDeTempo);

    log.info("sender.sendReceive(geraCertOtimizado);");
    // blocking call to web service
    OMElement res = sender.sendReceive(geraCertOtimizado);
    System.out.println("res:" + res);
    try {
      ByteArrayInputStream bais = new ByteArrayInputStream(Base64
          .decode(res.getFirstElement().getText()));
      log.info("Parsing the certificate from the response");
      CertificateFactory cf = CertificateFactory.getInstance("X509");
      ocCert = (X509Certificate) cf.generateCertificate(bais);
      log.info("Certificate generated successfully!");
      log.fine("Printing the optimized certificate");
      log.fine(ocCert.toString());
    } catch (CertificateException cex) {
      // an error occurred, print the error message.
      log
          .severe("Service returned an error message. \nMessage returned: "
              + new String(Base64.decode(res
                  .getFirstElement().getText())));
    }
    log.info("returned from call");
```

```
  } catch (AxisFault af) {
    af.printStackTrace();
  } catch (CertificateEncodingException e) {
    e.printStackTrace();
  }
  return ocCert;

}

/**
 * Creates the content and the SOAP envelope for the
 * {@link ClienteWS#iniciaClienteSeparado()} method.
 *
 * @param certUser
 *            The user's certificate.
 * @return the OMElement envelope
 * */
private OMElement geraPayload(X509Certificate certUser,
    List<X509Certificate> certsIntermediarios, List<X509CRL> crls,
    List<TimeStampToken> ts) throws CertificateEncodingException {

  byte[] certificadoUsuario = certUser.getEncoded();

  ByteArrayOutputStream baos = new ByteArrayOutputStream();
  for (X509Certificate c : certsIntermediarios) {
    try {
      baos.write(c.getEncoded());
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
  byte[] certificadosIntermediarios = baos.toByteArray();

  byte[] listaCRLs = null;
  if (crls != null && crls.size() > 0) {
    baos = new ByteArrayOutputStream();
    for (X509CRL crl : crls) {
      try {
        baos.write(crl.getEncoded());
      } catch (CRLException e) {
        e.printStackTrace();
      } catch (IOException e) {
        e.printStackTrace();
      }
```

```
    }
    listaCRLs = baos.toByteArray();
  }

  byte[] listaCarimbos = null;
  if (crls != null && crls.size() > 0) {
    baos = new ByteArrayOutputStream();
    for (TimeStampToken t : ts) {
      try {
        baos.write(t.getEncoded());
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
    listaCarimbos = baos.toByteArray();
  }

  return geraPayload(certificadoUsuario, certificadosIntermediarios,
      listaCRLs, listaCarimbos);
}

/**
 * Creates the content and the SOAP envelope for the
 * {@link ClienteWS#iniciaClienteSeparado()} method.
 * */
private OMElement geraPayload(byte[] certUser, byte[] certsIntermediarios,
    byte[] crls, byte[] ts) {

  SOAPFactory fac = OMAbstractFactory.getSOAP11Factory();
  OMNamespace omNs = fac.createOMNamespace("http://normal.jocca.ws", "");
  OMElement geraCertOtimizado = fac.createOMElement("geraCertOtimizado",
      omNs);

  /*
   * OMElement hashEle = fac.createOMElement("hash", omNs);
   * hashEle.setText("hashomg warhgarbl");
   *
   * OMElement sigEle = fac.createOMElement("sig", omNs);
   * sigEle.setText("Assinado: Gumercindo Costa e Silva");
   */

  OMElement certUserEle = fac.createOMElement("certUser1", omNs);
  certUserEle.setText(Util.converteEmBase64(certUser));
```

```
  OMElement certsIntermediariosEle = fac.createOMElement(
      "certsIntermediarios", omNs);
  certsIntermediariosEle.setText(Util
      .converteEmBase64(certsIntermediarios));

  OMElement CRLsEle = fac.createOMElement("crls", omNs);

  CRLsEle.setText(Util.converteEmBase64(crls));

  OMElement tspEle = fac.createOMElement("timestamps", omNs);
  /* sem carimbos de tempo por enquanto */
  // tspEle.setText(Util.converteEmBase64(ts));
  tspEle.setText("");

  /*
   * adiciona os argumentos à chamada do método IMPORTANTE: A ordem em que
   * os nós são adicionados ao envelope importa - é a ordem dos argumentos
   * do método
   */

  geraCertOtimizado.addChild(certUserEle);
  geraCertOtimizado.addChild(certsIntermediariosEle);
  geraCertOtimizado.addChild(CRLsEle);
  geraCertOtimizado.addChild(tspEle);

  return geraCertOtimizado;
}

/**
 * Gera o conteúdo e o envelope a ser enviado ao web service, contendo o CMS
 * */
private OMElement geraPayloadCMS(byte[] cms, byte[] hash) {

  SOAPFactory fac = OMAbstractFactory.getSOAP11Factory();
  OMNamespace omNs = fac.createOMNamespace(nameSpace, "");
  OMElement geraCertOtimizado = fac.createOMElement(
      "geraCertOtimizadoCMS", omNs);

  OMElement cmsEle = fac.createOMElement("cms1", omNs);
  cmsEle.setText(Util.converteEmBase64(cms));

  OMElement hashEle = fac.createOMElement("hash", omNs);
  hashEle.setText(Util.converteEmBase64(hash));
```

```
  geraCertOtimizado.addChild(cmsEle);
  geraCertOtimizado.addChild(hashEle);

  return geraCertOtimizado;
}

/**
 * Gera o conteúdo e o envelope a ser enviado ao web service, contendo o CMS
 * */
private OMElement geraPayloadRetornaCertACCO() {

  SOAPFactory fac = OMAbstractFactory.getSOAP11Factory();
  OMNamespace omNs = fac.createOMNamespace(nameSpace, "");
  OMElement geraCertOtimizado = fac.createOMElement(
      "retornaCertificadoACCO", omNs);

  return geraCertOtimizado;
}

/**
 * Reads the configuration file from the disk.
 *
 * @throws FileNotFoundException
 *             if the configuration file can't be found.
 * */
private boolean leConfiguracaoDoDisco() throws FileNotFoundException {
  boolean retorno = false, epr = false;
  // acco-client.config
  File f = new File("acco-client.config");
  if (f.exists() & f.canRead()) {
    Scanner s = new Scanner(f);

    String caminho = "", caminho_epr = "";

    while (s.hasNext()) {
      String linha = s.nextLine();
      if (!linha.startsWith("#")) {// ignores comments (lines starting
                    // with #)
        if (linha.startsWith("servidor=")) {
          caminho_epr = linha.substring(9);
          epr = true;
        }
      }
    }
```

```
      retorno = epr;
      if (retorno) {// all mandatory configs were found
        targetEPR = new EndpointReference(caminho_epr);
        log.info("targetEndPointReference:" + caminho_epr);
      }
    } else {
      log.severe("configuration file couldn't be found/read. aborting.");
    }
    return retorno;
  }
}
```

### 7.2.2 Classe: EntradaWS

```
package ws.jocca.normal;

import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.KeyFactory;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.PrivateKey;
import java.security.Signature;
import java.security.SignatureException;
import java.security.cert.CRL;
import java.security.cert.CRLException;
import java.security.cert.CertPath;
import java.security.cert.CertPathBuilderException;
import java.security.cert.CertStore;
import java.security.cert.CertStoreException;
import java.security.cert.Certificate;
import java.security.cert.CertificateEncodingException;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.cert.TrustAnchor;
import java.security.cert.X509CRL;
import java.security.cert.X509CRLSelector;
import java.security.cert.X509CertSelector;
```

```
import java.security.cert.X509Certificate;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.PKCS8EncodedKeySpec;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Date;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Scanner;
import java.util.Set;
import java.util.logging.Logger;

import org.bouncycastle.asn1.ASN1OctetString;
import org.bouncycastle.asn1.cms.Attribute;
import org.bouncycastle.asn1.cms.AttributeTable;
import org.bouncycastle.asn1.cms.CMSAttributes;
import org.bouncycastle.cms.CMSException;
import org.bouncycastle.cms.CMSSignedData;
import org.bouncycastle.cms.SignerId;
import org.bouncycastle.cms.SignerInformation;
import org.bouncycastle.cms.SignerInformationStore;
import org.bouncycastle.jce.provider.BouncyCastleProvider;
import org.bouncycastle.util.encoders.Base64;

import sun.security.timestamp.TimestampToken;
import ws.jocca.normal.crypto.Crypto;
import ws.jocca.normal.lib.COException;
import ws.jocca.normal.lib.Util;

/**
 * This class contains the methods exposed by the web service.
 * */
public class EntradaWS {

  CertificateFactory cf;
  static Logger log = Logger.getLogger(EntradaWS.class.getName());
  static String pathToICP = ""; // conf

  public EntradaWS() throws Exception {
    if (!leConfiguracaoDoDisco()) {
      log.severe("Arquivo de configuração não encontrado, saindo.");
      throw new Exception(
          "ERRO: Arquivo de configuração não encontrado, saindo.");
```

```
  }
}

/**
 * Method exposed by the Web Service. Takes a CMS file containing the user's
 * certificate and all certificates needed to build it's certification path,
 * and the signature of a document. May include CRLs and time stamps. The
 * additional hash parameter is the hash of the signed document, to be
 * checked against the hash in the signature. If the certificate is valid
 * and the hashes match, an optimized certificate is issued and returned
 * with Base64 encoding.
 *
 * @param cms
 *            Base-64 encoded CMS
 * @param hash
 *            The Base-64 encoded hash of the signed file
 * @return The optimized certificate, encoded with Base64.
 * @throws Exception
 */
public String geraCertOtimizadoCMS(String cms1, String hash) {

  ByteArrayInputStream bais = new ByteArrayInputStream(Base64
      .decode(cms1));
  String retorno = "";
  X509Certificate ocCert = null;
  boolean correto = false;
  CMSSignedData cms = null;
  try {
    // cf=CertificateFactory.getInstance("X509", new
    // BouncyCastleProvider());
    cf = CertificateFactory.getInstance("X509");
    cms = new CMSSignedData(bais);
    log.info("Obtaining certificates and CRLs from the CMS");
    CertStore certsAndCRLs = cms.getCertificatesAndCRLs("Collection",
        "SUN");
    X509CertSelector selector = new X509CertSelector();
    Collection<X509Certificate> listaCertsIntermediarios =
    (Collection<X509Certificate>) certsAndCRLs.getCertificates(selector);
    X509CRLSelector selectorCRL = new X509CRLSelector();
    Collection<CRL> listaCRLs = (Collection<CRL>) certsAndCRLs
        .getCRLs(selectorCRL);

    X509Certificate certUser;
    SignerInformationStore siStore = cms.getSignerInfos();
```

```
ArrayList<SignerInformation> signers = (ArrayList<SignerInformation>) siStore
    .getSigners();
SignerId sigIDUSer = signers.get(0).getSID();
Set<Certificate> temp = (Set<Certificate>) certsAndCRLs
    .getCertificates(sigIDUSer);
List<Certificate> assinantes = new ArrayList<Certificate>(temp);
certUser = (X509Certificate) assinantes.get(0);

log.info("checking the signature in the CMS");
// checks if the signature is valid
CertStore certs = cms.getCertificatesAndCRLs("Collection",
    new BouncyCastleProvider());
Iterator<SignerInformation> it = signers.iterator();
int verified = 0, failed = 0;

log.finer("number of signers: "
    + cms.getSignerInfos().getSigners().size());
while (it.hasNext()) {
  log.info("Checando assinaturas");
  SignerInformation signer = (SignerInformation) it.next();
  Collection<X509Certificate> certCollection = (Collection<X509Certificate>) certs
      .getCertificates(signer.getSID());
  Iterator<X509Certificate> certIt = certCollection.iterator();
  X509Certificate cert = certIt.next();
  if (signer.verify(cert, new BouncyCastleProvider()))
    verified++;
  else
    failed++;
}

if (verified > 0) {
  log
      .info("valid signature(s)! verified signatures: "
          + verified);
  if (failed > 0)
    log.warning("some signatures failed checking: " + failed);
} else {
  throw new COException("ERROR: no valid signatures found!");
}

// compares the signed hash with the hash passed via parameter
// checks only the first signer,for now
Iterator<SignerInformation> it1 = siStore.getSigners().iterator();
SignerInformation signer = it1.next();
```

```
  AttributeTable table = signer.getSignedAttributes();
  Attribute hashAttr = table.get(CMSAttributes.messageDigest);
  byte[] hash2 = ((ASN1OctetString) hashAttr.getAttrValues()
      .getObjectAt(0)).getOctets();
  if (MessageDigest.isEqual(hash2, Base64.decode(hash))) {
    log.info("Hashes match");
  } else {
    throw new COException(
        "Error validating hashes: the hashes don't match.");
  }

  // Crypto
  PrivateKey privkey = lerChaveACCO();
  log.fine("OCCA's private key read successfully");
  X509Certificate certRoot = lerCertACRaiz();
  TrustAnchor t = new TrustAnchor(certRoot, null);
  Set<TrustAnchor> trust = new HashSet<TrustAnchor>();
  trust.add(t);
  X509Certificate certACCO = lerCertACCO();
  listaCertsIntermediarios.add(certACCO);
  log.info("Instantiating Crypto");
  Crypto crypto = new Crypto(certACCO, privkey, trust);

  // builds the certification path
  log.info("Building the certification path");
  // CertPath cp = crypto.validateCert(certUser,
  // listaCertsIntermediarios, listaCRLs, new Date());
  CertPath cp = crypto.constroiCaminho(certUser,
      listaCertsIntermediarios, listaCRLs, new Date());
  log.info("Issuing certificate");
  ocCert = crypto.issueCert(certUser, cp, hash);
  // prints the certificate
  // log.info(ocCert.toString());

  correto = true;

} catch (CMSException e) {
  retorno = "ERROR invalid parameter.";
  e.printStackTrace();
} catch (NoSuchProviderException e) {
  e.printStackTrace();
} catch (NoSuchAlgorithmException e) {
  retorno = "Internal error";
  log.severe("Invalid algorithm when loading OCCA's private key.");
```

```
  // log.severe("Erro na especificação do algoritmo para
  //importação da chave privada da ACCO.");
  e.printStackTrace();
} catch (InvalidKeySpecException e) {
  // log.severe("Erro no formato da chave da ACCO.");
  log
      .severe("Error loading the OCCA's private key: unknown key "+
"specification.");
  e.printStackTrace();
} catch (IOException e) {
  retorno = "Internal error.";
  // log.severe("Erro na leitura da chave privada da ACCO/certificado
  //da ACCO/certificado da AC-Raiz");
  log
      .severe("Error reading the OCCA's private key/OCCA's certificate/"+
  "Root CA's certificate.");
  e.printStackTrace();
} catch (CertStoreException e) {
  // retorno="ERRO Certificados e/ou CRLs inválidos";
  retorno = "ERROR: invalid certificates and/or CRLs.";
  // log.warning("Erro na construção do CertStore usando os certificados
  //do CMS(Crypto)");
  log
      .warning("Error when creating the CertStore using certificates from the CMS");
  e.printStackTrace();
} catch (InvalidAlgorithmParameterException e) {
  retorno = "Internal Error.";
  e.printStackTrace();
} catch (CertPathBuilderException e) {
  retorno = "Error: unable to build certification path.";
  e.printStackTrace();
} catch (CertificateException e) {
  e.printStackTrace();
} catch (CRLException e) {
  e.printStackTrace();
} catch (COException e) {
  retorno = e.getMessage();
  log.info(retorno);
  e.printStackTrace();
} finally {
  /* Response */
  if (correto) {// if the OC was successfully issued, encode it with
          // Base64 and send it back
    try {
```

```
        retorno = Util.converteEmBase64(ocCert.getEncoded());
      } catch (CertificateEncodingException e) {
        log.severe("Crypto generated a faulty certificate");
        retorno = "Internal error.";
        e.printStackTrace();
      }
    } else {// An error occurred, return the error message encoded with
        // Base64
      retorno = Util.converteEmBase64(retorno.getBytes());
    }
  }
  return retorno;
}

/**
 * Method exposed by the web service. This method does the same as
 * {@link EntradaWS#geraCertOtimizadoCMS(String, String)}, but it takes the
 * components separately. All parameters need to be Base64-encoded.
 *
 * @param certUser1
 *            User's X509 certificate
 * @param certsIntermediarios
 *            Certificates needed to build the certification path,
 *            concatenated (in PEM format).
 * @param crls
 *            Any number of CRLs, concatenated (in PEM format).
 * @param sig
 *            The document's signature
 * @param timestamps
 *            Any number of time stamps required to validate the certificate
 *            or signature.
 * @param hash
 *            The hash of the signed document
 */
@SuppressWarnings("unchecked")
@Deprecated
public String geraCertOtimizado(String certUser1,
    String certsIntermediarios, String crls, String sig,
    String timestamps, String hash) {

  String resposta = "";
  X509Certificate certUser, certRoot, ocCert = null;
  List<X509Certificate> listaCertsIntermediarios;
  List<CRL> listaCRLs;
```

```
Collection<TimestampToken> listaCarimbos;

boolean retorno = false;
/* Instantiating certificates, CRLs and timestamps */
try {
  log.info("Getting instance of CertificateFactory");
  cf = CertificateFactory.getInstance("X.509");
  log.info("Reading Root CA's Certificate");
  certRoot = lerCertACRaiz();

  log.info("Loading the user's certificate");
  byte[] temp = Base64.decode(certUser1);
  ByteArrayInputStream bais = new ByteArrayInputStream(temp);
  certUser = (X509Certificate) cf.generateCertificate(bais);
  log.fine("User's certificate loaded successfully");

  log.info("Loading other certificates");
  temp = Base64.decode(certsIntermediarios);
  bais = new ByteArrayInputStream(temp);
  listaCertsIntermediarios = new ArrayList<X509Certificate>(
      (Collection<? extends X509Certificate>) cf
          .generateCertificates(bais));
  log.info("Loaded " + listaCertsIntermediarios.size()
      + "certificates");

  if (!crls.equals("")) {
    log.info("Loading CRLs");
    temp = Base64.decode(crls);
    bais = new ByteArrayInputStream(temp);
    listaCRLs = new ArrayList<CRL>(
        (Collection<? extends X509CRL>) cf.generateCRLs(bais));
    log.info("Loaded " + listaCRLs.size() + "CRLs");
  } else {
    log.info("No CRLs sent");
    listaCRLs = new ArrayList<CRL>();
  }

  // for the moment, timestamps are not used.

  // checks the signature

  if (validaAssinatura(hash, sig, certUser)) {
    log.finer("Signature verified");
  } else {
```

```
      throw new COException("The signature verification failed.");
    }

    // Crypto
    PrivateKey privkey = lerChaveACCO();
    log.fine("OCCA's private key loaded successfully");
    TrustAnchor t = new TrustAnchor(certRoot, null);
    Set<TrustAnchor> trust = new HashSet<TrustAnchor>();
    trust.add(t);
    X509Certificate certACCO = lerCertACCO();
    log.info("Instantiating Crypto");
    Crypto crypto = new Crypto(certACCO, privkey, trust);

    // builds the certification path
    log.info("building certification path");
    CertPath cp = crypto.constroiCaminho(certUser,
        listaCertsIntermediarios, listaCRLs, new Date());
    log.info("issuing optimized certificate");
    ocCert = crypto.issueCert(certUser, cp, hash);
    retorno = true;
    log.info(ocCert.toString());

  } catch (CertificateException e) {
    e.printStackTrace();
  } catch (IOException e) {
    log
        .severe("Error loading the OCCA's private key/Root CA's private "+
     "key/OCCA's certificate/current Novomodo proof from the disk.");
    resposta = "Internal error.";
    e.printStackTrace();
  } catch (InvalidKeySpecException e) {
    log
        .severe("Error loading the OCCA's private key/Root CA's private key.");
    e.printStackTrace();
  } catch (NoSuchAlgorithmException e) {
    log.severe("Invalid algorithm specified.");
    resposta = "Internal error.";
    e.printStackTrace();
  } catch (InvalidAlgorithmParameterException e) {
    log.severe("Invalid algorithm parameter(s) specified.");
    resposta = "Internal error.";
    e.printStackTrace();
  } catch (CertPathBuilderException e) {
    log.info("Unable to build certification path.");
```

```
      resposta = "Error: unable to build certification path.";
      e.printStackTrace();
    } catch (NoSuchProviderException e) {
      log.severe("Error: no such provider.");
      resposta = "Internal error.";
      e.printStackTrace();
    } catch (CRLException e) {
      log.info("ERRO na leitura das CRLs");
      resposta = "Error: unable to load one or more CRLs";
      e.printStackTrace();
    } catch (CertStoreException e) {
      resposta = "Internal error.";
      e.printStackTrace();
    } catch (SignatureException e) {
      log.info("Invalid or corrupted signature.");
      resposta = "Invalid or corrupted signature.";
      e.printStackTrace();
    } catch (COException e) {
      log.info(e.getMessage());
      resposta = e.getMessage();
      e.printStackTrace();
    }

    /* response */
    if (retorno) {// if the OC was successfully issued, encode it with
            // Base64 and send it back
      try {
        resposta = Util.converteEmBase64(ocCert.getEncoded());
      } catch (CertificateEncodingException e) {
        log.severe("Crypto generated a faulty certificate");
        resposta = "Internal error.";
        e.printStackTrace();
      }
    } else {// An error occurred, return the error message encoded with
        // Base64
      resposta = Util.converteEmBase64(resposta.getBytes());
    }
    return resposta;
  }

  /**
   * Method exposed via web service, returns the OCCA certificate used for
   * signing the optimized certificates.
   *
```

```
 * @return The Base64-encoded OCCA's certificate
 * */
public String retornaCertificadoACCO() {
  log.info("returning the OCCA's certificate");
  String retorno = "";
  try {
    retorno = new String(Base64.encode(lerCertACCO().getEncoded()));
  } catch (CertificateEncodingException e) {
    e.printStackTrace();
  } catch (CertificateException e) {
    e.printStackTrace();
  } catch (FileNotFoundException e) {
    e.printStackTrace();
  }
  return retorno;
}

/*
 * Auxiliary methods
 */

/**
 * Reads the configuration file from the disk.
 *
 * @throws FileNotFoundException
 *             if the configuration file can't be found.
 * */
private boolean leConfiguracaoDoDisco() throws FileNotFoundException {
  boolean retorno = false;
  // acco-server.config
  File f = new File("acco-server.config");
  if (f.exists() & f.canRead()) {
    Scanner s = new Scanner(f);

    String caminho = "";

    while (s.hasNext()) {
      String linha = s.nextLine();
      if (!linha.startsWith("#")) {// ignores comments (lines starting
                    // with #)
        if (linha.startsWith("pathToICP")) {
          caminho = linha.substring(10);
          retorno = true;
          break;
```

```
        }
      }
    }
    if (!caminho.endsWith("/")) {
      pathToICP = caminho + "/";
    } else {
      pathToICP = caminho;
    }
    log.info("pathToICP:" + pathToICP);
  } else {
    log.severe("configuration file couldn't be found/read. aborting.");
  }
  return retorno;
}

/**
 * Reads the OCCA's private key from the disk. The file must be in
 * unencrypted PKCS8 format.
 *
 * @return The OCCA's PrivateKey
 *
 * @throws IOException
 *             If the file can't be read.
 * @throws InvalidKeySpecException
 *             If the key encoding isn't valid.
 * @throws NoSuchAlgorithmException
 *             An invalid algorithm was given to KeyFactory
 * */
private PrivateKey lerChaveACCO() throws IOException,
    NoSuchAlgorithmException, InvalidKeySpecException {
  PrivateKey privkey = null;
  log.info("Reading the OCCA's private key");
  byte[] bytesChave = Util.leArquivoByteArray(pathToICP
      + "acco-nivel1/privkey-pk8.pem");
  PKCS8EncodedKeySpec pkcs8 = new PKCS8EncodedKeySpec(bytesChave);
  KeyFactory factory = KeyFactory.getInstance("RSA");
  privkey = factory.generatePrivate(pkcs8);
  return privkey;
}

/**
 * Reads the OCCA's certificate from the disk.
 *
 * @throws FileNotFoundException
```

```
 * @throws CertificateException
 */
private X509Certificate lerCertACCO() throws CertificateException,
    FileNotFoundException {
  X509Certificate retorno = null;
  log.info("Reading the OCCA's certificate");

  retorno = Util.leCert(pathToICP + "acco-nivel1/cert.pem");

  log.fine("Leu certificado da ACCO");
  return retorno;
}

/**
 * Reads the Root CA's certificate from the disk
 *
 * @throws FileNotFoundException
 * @throws CertificateException
 */
private X509Certificate lerCertACRaiz() throws CertificateException,
    FileNotFoundException {
  X509Certificate retorno = null;
  log.info("Reading the Root CA's certificate");
  retorno = Util.leCert(pathToICP + "ac-raiz/cert.pem");
  log.fine("Root CA certificate read successfully");
  return retorno;
}

/**
 * Validates the signature agains the hash sent in the request and the
 * user's certificate.
 *
 * @param hashStr
 *            The document's hash
 * @param sigStr
 *            The signature
 * @param certUser
 *            The user's certificate, who created the signature
 *
 * @return <code>true</code> if the signature was verified,
 *         <code>false</code> if not.
 *
 * @throws SignatureException
 *
```

```
     * @see Signature#verify(byte[])
     * */
    private boolean validaAssinatura(String hashStr, String sigStr,
        X509Certificate certUser) throws SignatureException {
      Signature assinatura;
      boolean retorno = false;
      try {

        assinatura = Signature.getInstance(certUser.getSigAlgName(),
            new BouncyCastleProvider());
        assinatura.initVerify(certUser);

        byte[] hash = Base64.decode(hashStr);
        byte[] sig = Base64.decode(sigStr);

        assinatura.update(hash);
        retorno = assinatura.verify(sig);

      } catch (NoSuchAlgorithmException e) {
        log.severe("Algoritmo inválido!");
        e.printStackTrace();
      } catch (InvalidKeyException e) {
        log.severe("Chave inválida!");
        e.printStackTrace();
      }

      return retorno;
    }

}
```

### 7.2.3 Classe: Util

```
package ws.jocca.normal.lib;

import java.io.ByteArrayOutputStream;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.security.cert.CRLException;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
```

```
import java.security.cert.X509CRL;
import java.security.cert.X509Certificate;

import javax.activation.FileDataSource;

public class Util {

  /**
   * Reads a file from the disk, returning it as a string.
   *
   * @param caminhoDoArquivo
   *            The path of the file to be read.
   * @return The contents of the file as a string.
   * @throws IOException
   *             if the file can't be read.
   * */
  public static String leArquivo(String caminhoDoArquivo)
    throws IOException {
    String retorno = "";

    File f = new File(caminhoDoArquivo);
    int bytesLidos = 0;

    if (f.exists()) {
      ByteArrayOutputStream bytes = new ByteArrayOutputStream();
      FileDataSource fds = new FileDataSource(f);
      byte[] temp = new byte[64];
      boolean cabou = false;
      DataInputStream dis = new DataInputStream(fds.getInputStream());
      while (!cabou) {
        bytesLidos = 0;
        temp = new byte[64];
        bytesLidos = dis.read(temp);
        if (bytesLidos > 0) {
          bytes.write(temp, 0, bytesLidos);
        } else {
          cabou = true;
        }
      }

      retorno = new String(bytes.toByteArray());
    } else
      throw new IOException("arquivo " + caminhoDoArquivo
          + " não existe.");
```

```
  return retorno;
}

/**
 * Reads a file from the disk, and tries to create a X509Certificate object
 * from its contents.
 *
 * @param caminhoDoArquivo
 *            The path of the file to be read.
 * @return The X509Certificate
 * @throws CertificateException
 *             if the file didn't contain a valid encoded certificate, or if
 *             the file was corrupted.
 * @throws FileNotFoundException
 *             if the file couldn't be found.
 * */
public static X509Certificate leCert(String caminhoDoArquivo)
    throws CertificateException, FileNotFoundException {
  X509Certificate retorno = null;

  FileInputStream fis = new FileInputStream(caminhoDoArquivo);
  CertificateFactory cf = CertificateFactory.getInstance("X.509");
  retorno = (X509Certificate) cf.generateCertificate(fis);

  return retorno;
}

/**
 * Reads a file from the disk, and tries to create a X509CRL object from its
 * contents.
 *
 * @param caminhoDoArquivo
 *            The path of the file to be read.
 * @return The X509CRL
 * @throws CertificateException
 * @throws FileNotFoundException
 *             if the file couldn't be found.
 * @throws CRLException
 *             if the file didn't contain a valid CRL, or the file was
 *             corrupted.
 * */
public static X509CRL leCRL(String caminhoDoArquivo)
    throws FileNotFoundException, CertificateException, CRLException {
  X509CRL retorno = null;
```

```
  FileInputStream fis = new FileInputStream(caminhoDoArquivo);
  CertificateFactory cf = CertificateFactory.getInstance("X.509");
  retorno = (X509CRL) cf.generateCRL(fis);

  return retorno;
}

/**
 * Reads a file from the disk, and returns it as a byte array.
 *
 * @param caminhoDoArquivo
 *            The path of the file to be read.
 * @return The byte array.
 * @throws IOException
 *             if the file couldn't be read.
 * */
public static byte[] leArquivoByteArray(String caminhoDoArquivo)
    throws IOException {
  byte[] retorno = new byte[1];
  File f = new File(caminhoDoArquivo);
  int bytesLidos = 0;

  if (f.exists()) {
    ByteArrayOutputStream bytes = new ByteArrayOutputStream();

    FileInputStream fis = new FileInputStream(f);
    bytesLidos = fis.read();
    while (bytesLidos != -1) {
      bytes.write(bytesLidos);
      bytesLidos = fis.read();
    }
    retorno = bytes.toByteArray();
  } else
    throw new IOException("arquivo " + caminhoDoArquivo
        + " não existe.");
  return retorno;
}

/**
 * Encodes a given string with Base64 encoding.
 *
 * @param dados
 *            The string
```

```
   * @return the Base64-encoded string as a string.
   * */
  public static String converteEmBase64(String dados) {
    return new String(org.bouncycastle.util.encoders.Base64.encode(dados
        .getBytes()));
  }

  /**
   * Encodes a given byte array with Base64 encoding.
   *
   * @param dados
   *            The byte array
   * @return the Base64-encoded byte array as a string
   * */
  public static String converteEmBase64(byte[] dados) {
    return new String(org.bouncycastle.util.encoders.Base64.encode(dados));
  }

}
```

### 7.2.4 Classe: COException

```
 package ws.jocca.normal.lib;

public class COException extends Exception {

  /**
   *
   */
  private static final long serialVersionUID = 1L;

  public COException(String msg){
    super(msg);
  }
}
```

### 7.2.5 Classe: Crypto

```
 package ws.jocca.normal.crypto;

import java.io.IOException;
import java.math.BigInteger;
```

```
import java.security.InvalidAlgorithmParameterException;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.PrivateKey;
import java.security.cert.CRL;
import java.security.cert.CRLException;
import java.security.cert.CertPath;
import java.security.cert.CertPathBuilder;
import java.security.cert.CertPathBuilderException;
import java.security.cert.CertStore;
import java.security.cert.CertStoreException;
import java.security.cert.CertificateException;
import java.security.cert.CollectionCertStoreParameters;
import java.security.cert.PKIXBuilderParameters;
import java.security.cert.PKIXCertPathBuilderResult;
import java.security.cert.TrustAnchor;
import java.security.cert.X509CertSelector;
import java.security.cert.X509Certificate;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Date;
import java.util.List;
import java.util.Set;

import org.bouncycastle.asn1.x509.BasicConstraints;
import org.bouncycastle.jce.provider.BouncyCastleProvider;
import org.bouncycastle.x509.X509V3CertificateGenerator;

import ws.jocca.normal.crypto.ValidationTimeExtension.timeSourceOptions;
import ws.jocca.normal.lib.Util;

public class Crypto {

  protected X509Certificate myCert;
  protected PrivateKey privKey;
  protected Set<TrustAnchor> trusted;

  public Crypto(X509Certificate cert, PrivateKey privKey,
      Set<TrustAnchor> trusted) {
    this.myCert = cert;
    this.privKey = privKey;
    this.trusted = trusted;
  }
```

```
/**
 * Issues a new optimized certificate for the given certificate.
 *
 * @param aCert
 *            The user's certificate.
 * @param certPath
 *            The certification path for the user's certificate.
 * @param hash
 *            The hash of the signed document.
 *
 * @throws IOException
 *             if the program is unable to read the current Novomodo Proof
 * */
public X509Certificate issueCert(X509Certificate aCert, CertPath certPath,
    String hash) throws IOException {
  X509V3CertificateGenerator generator = new X509V3CertificateGenerator();
  X509Certificate ret = null;
  generator.setIssuerDN(this.myCert.getSubjectX500Principal());
  generator.setSubjectDN(aCert.getSubjectX500Principal());
  Date data = new Date();
  generator.setNotAfter(data);
  generator.setNotBefore(data);
  generator.setSerialNumber(new BigInteger("16"));
  generator.setPublicKey(aCert.getPublicKey());
  generator.setSignatureAlgorithm("1.2.840.113549.1.1.5");

  BasicConstraints bc = new BasicConstraints(false);
  generator.addExtension("2.5.29.19", true, bc.getDEREncoded());
  // novomodoProof extension
  NovomodoProofExtension novomodoProof = new NovomodoProofExtension(
      new String(this.getCurrentNovomodoProof()));
  generator.addExtension("0.0.0.0.0.0.1", false, novomodoProof
      .toASN1Object());
  // signeddocdigest extension
  SignedDocDigestExtension signeddocdigest = new SignedDocDigestExtension(
      false, "2.16.840.1.101.3.4.2.3", hash);
  generator.addExtension("0.0.0.0.0.0.2", false, signeddocdigest
      .toASN1Object());
  // validationtimeextension extension
  ValidationTimeExtension validationtime = new ValidationTimeExtension(
      timeSourceOptions.OCCA_CLOCK, new Date());
  generator.addExtension("0.0.0.0.0.0.3", false, validationtime
      .getDEREncoded());
```

```
  // novomodotargets extension

  // NovomodoTargetsExtension novomodotargets = new
  // NovomodoTargetsExtension(true, "2.16.840.1.101.3.4.2.3",
  // "validationtarget".getBytes(), "revocationtarget".getBytes(), 5);
  // NovomodoTargetsExtension novomodotargets = new
  // NovomodoTargetsExtension("1.3.14.3.2.26",
  // "Ucu6KX7MiNc6J/ISys4XWHWT1Zk=", "jCDs/NGohHgrdSGN8CC/fEbw1zM=",
  // 86400);
  // generator.addExtension("0.0.0.0.0.5", false,
  // novomodotargets.toASN1Object());//pra fechar com o SimpleCA
  try {
    // OriginalCertPathExtension extension
    // OriginalCertPathExtension ocpe = new
    // OriginalCertPathExtension(false, certPath);
    // generator.addExtension("0.0.0.0.0.6", false,
    // ocpe.toASN1Object());
    ret = generator.generate(this.privKey);
  } catch (Exception e) {
    e.printStackTrace();
  }

  return ret;
}

/**
 * Gets the up-to-date Novomodo Proof (currently, it reads a file from the
 * disk, which contains the Base64-encoded proof).
 *
 * @return byte array containing the base64-encoded proof.
 * */
private byte[] getCurrentNovomodoProof() {
  byte[] retorno = null;
  try {
    retorno = Util.leArquivoByteArray("provaAtual.txt");
  } catch (IOException e) {
    System.out.println("Error when reading the current proof.");
    e.printStackTrace();
  }
  return retorno;
}

/**
 * Tries to build the certification path for the target certificate, using
```

```
 * the provided certificates and CRLs. Different implementation of
 * {@link Crypto#constroiCaminho(X509Certificate, Collection, Collection, Date)}
 * , the other method is used.
 *
 * @param target
 *            The target certificate
 * @param cas
 *            Collection containing the certificates that comprise the
 *            target's certification path. The list may contain other
 *            unrelated certificates.
 * @param crls
 *            Collection of CRLs to be checked.
 * @param time
 *            The date at which the certificate will be checked to be valid.
 *            If null, the current time will be used.
 * @returns The CertPath object if the certification path was built
 *          successfully. A CertPathBuilderException is thrown if the
 *          process fails.
 *
 * @throws NoSuchAlgorithmException
 * @throws InvalidAlgorithmParameterException
 * @throws CertPathBuilderException
 * @throws NoSuchProviderException
 *
 * */
public CertPath validateCert(X509Certificate target,
    Collection<X509Certificate> cas, Collection<CRL> crls, Date time)
    throws NoSuchAlgorithmException,
    InvalidAlgorithmParameterException, CertPathBuilderException,
    NoSuchProviderException {
  X509CertSelector certSelector = new X509CertSelector();
  PKIXBuilderParameters builderParams = null;
  CertPathBuilder certPathBuilder = null;
  CertPath certPath = null;
  Collection casAndCrls = new ArrayList();

  // puts CRLs and certificates together in a list
  casAndCrls.addAll(cas);
  casAndCrls.add(crls);

  // Set target certificate
  certSelector.setCertificate(target);
  // certSelector.setIssuer(target.getIssuerX500Principal());
  // certSelector.setSubjectPublicKey(target.getPublicKey());
```

```
  // certSelector.setSerialNumber(target.getSerialNumber());

  // creates a CertStore with the CRLs and certificates composing the
  // certification path
  CertStore cs = CertStore.getInstance("Collection",
      new CollectionCertStoreParameters(casAndCrls));

  // Creates object that defines the parameters for the creation and
  // validation of the certification path
  builderParams = new PKIXBuilderParameters(this.trusted, certSelector);
  builderParams.addCertStore(cs);

  // If there are CRLs, then check the revocation status
  if (crls != null && crls.size() > 0) {
    builderParams.setRevocationEnabled(true);
  } else {
    builderParams.setRevocationEnabled(false);
  }

  if (time != null) {
    builderParams.setDate(time);
  }

  // attempts to create certification path
  certPathBuilder = CertPathBuilder.getInstance(CertPathBuilder
      .getDefaultType(), "SUN");
  // certPathBuilder =
  // CertPathBuilder.getInstance(CertPathBuilder.getDefaultType(), new
  // BouncyCastleProvider());
  certPath = certPathBuilder.build(builderParams).getCertPath();

  return certPath;
}

/**
 * Tries to build the certification path for the target certificate, using
 * the provided certificates and CRLs.
 *
 * @param target
 *            The target certificate
 * @param cas
 *            Collection containing the certificates that comprise the
 *            target's certification path. The list may contain other
 *            unrelated certificates.
```

```
 * @param crls
 *            Collection of CRLs to be checked.
 * @param time
 *            The date at which the certificate will be checked to be valid.
 *            If null, the current time will be used.
 * @returns The CertPath object if the certification path was built
 *          successfully. A CertPathBuilderException is thrown if the
 *          process fails.
 *
 * @throws CertPathBuilderException
 *             if the builder cannot build a valid path using the given
 *             certificates.
 * @throws NoSuchAlgorithmException
 * @throws NoSuchProviderException
 * @throws CertStoreException
 * @throws InvalidAlgorithmParameterException
 * @throws CertificateException
 * @throws CRLException
 * */
public CertPath constroiCaminho(X509Certificate target,
    Collection<X509Certificate> cas, Collection<CRL> crls, Date time)
    throws NoSuchAlgorithmException, NoSuchProviderException,
    CertStoreException, InvalidAlgorithmParameterException,
    CertPathBuilderException, CertificateException, CRLException {

  List listDeCertsECrls = new ArrayList();
  listDeCertsECrls.addAll(cas);
  listDeCertsECrls.addAll(crls);
  listDeCertsECrls.add(target);

  CollectionCertStoreParameters params = new CollectionCertStoreParameters(
      listDeCertsECrls);

  // CertStore store = CertStore.getInstance("Collection", params, "SUN");
  CertStore store = CertStore.getInstance("Collection", params,
      new BouncyCastleProvider());

  // build the path
  CertPathBuilder builder = CertPathBuilder.getInstance("PKIX", "SUN");
  // CertPathBuilder builder = CertPathBuilder.getInstance("PKIX", new
  // BouncyCastleProvider());
  X509CertSelector endConstraints = new X509CertSelector();

  // endConstraints.setSerialNumber(userCert.getSerialNumber());
```

```
    // endConstraints.setIssuer(userCert.getIssuerX500Principal().getEncoded());
    // endConstraints.setCertificate(target);
    endConstraints.setSubjectPublicKey(target.getPublicKey());

    PKIXBuilderParameters buildParams = new PKIXBuilderParameters(trusted,
        endConstraints);

    buildParams.addCertStore(store);
    if (time != null) {
      buildParams.setDate(time);
    } else {
      buildParams.setDate(new Date());
    }

    // If there are CRLs, then check the revocation status
    if (crls != null && crls.size() > 0) {
      buildParams.setRevocationEnabled(true);
    } else {
      buildParams.setRevocationEnabled(false);
    }
    PKIXCertPathBuilderResult result = (PKIXCertPathBuilderResult) builder
        .build(buildParams);
    CertPath path = result.getCertPath();

    return path;
  }

}
```