UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Anderson Luiz Silvério

ANÁLISE E IMPLEMENTAÇÃO DE UM PROTOCOLO DE GERENCIAMENTO DE CERTIFICADOS

Florianópolis

2011

Anderson Luiz Silvério

ANÁLISE E IMPLEMENTAÇÃO DE UM PROTOCOLO DE GERENCIAMENTO DE CERTIFICADOS

Trabalho de Conclusão de Curso submetido ao Curso de Ciências da Computação para a obtenção do Grau de Bacharel em Ciência da Computação.
Orientador: Jonathan Gehard Kohler, MSc.
Coorientador: Prof. Ricardo Felipe Custódio, Dr.

Florianópolis

2011

Anderson Luiz Silvério

ANÁLISE E IMPLEMENTAÇÃO DE UM PROTOCOLO DE GERENCIAMENTO DE CERTIFICADOS

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de "Bacharel em Ciência da Computação", e aprovado em sua forma final pelo Curso de Ciências da Computação.

Florianópolis, 15 de julho 2011.

Prof. Vitório Bruno Mazzola, Dr.
Coordenador

Banca Examinadora:

Jonathan Gehard Kohler, MSc.
Orientador

Prof. Ricardo Felipe Custódio, Dr.
Coorientador

Rogerio Bodemüller Junior, BSc.

Cristian Thiago Moecke, BSc.

Marcelo Carlomagno Carlos, MSc.

À minha família que tornou a realização deste trabalho possível.

AGRADECIMENTOS

Primeiramente gostaria de agradecer a todas as pessoas que me ajudaram de forma direta ou indireta na realização deste trabalho.

À minha família, em especial à minha mãe Jussara Zipperer e minha tia Iara Zipperer, pelo apoio e pelas oportunidades proporcionadas para eu chegar neste ponto.

Ao orientador deste trabalho, Jonathan Gehard Kohler, pelos ensinamentos e conselhos, não apenas relacionados a este trabalho mas ao longo dos últimos três anos, e pelas horas dedicadas à este trabalho.

Por fim, agradeço aos membros do LabSEC por sempre sanarem minhas dúvidas, além do auxílio nos mais diversos assuntos.

“O único lugar onde o sucesso vem antes do trabalho é no dicionário.” Vince Lombardi

RESUMO

Este trabalho tem como objetivo estudar e implementar um protocolo para a comunicação entre Autoridades Certificadoras e de Registro para o Sistema de Gerenciamento de Certificados Digitais da Infraestrutura de Chaves Públicas para Ensino e Pesquisa (SGCI). A versão 1.3.7 do SGCI possui um protocolo próprio para fazer a comunicação entre ACs e ARs, que não garante a integridade e autenticidade das mensagens trocadas entre AC e AR. Este trabalho propõe a implementação de um protocolo baseado no *Certificate Management Protocol* (CMP). Para tanto, foram analisadas e implementadas as mudanças necessárias ao SGCI para suportar tal protocolo, além do desenvolvimento de uma biblioteca para a geração e manipulação das mensagens do CMP em XML.

Palavras-chave: Criptografia, ICPEDU, Certificado Digital, x509, SGCI, Autoridade Certificadora, CMP.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

A sociedade moderna está convergindo para o uso de meios eletrônicos e *online* para a realização de tarefas cotidianas. Por exemplo, o uso do *internet banking* para pagar contas ao invés de ir pessoalmente ao banco. Com isso é possível reduzir o tempo gasto nestas tarefas por parte dos usuários e o custo em manter uma estrutura física por parte das empresas. Por outro lado, na ausência de contato visual entre as partes envolvidas em sistemas *online*, há a necessidade de mecanismos de autenticação. Também são necessários mecanismos de segurança que devem garantir a integridade dos dados compartilhados entre usuário e sistema, autenticar as partes envolvidas, entre outros requisitos. A certificação digital tem sido utilizada de modo a satisfazer tais requisitos.

Certificados digitais ligam uma pessoa ou entidade à um par de chaves. Com o par de chaves é possível realizar operações, como a assinatura digital, que garantem a autenticidade, integridade, confidencialidade e irretratabilidade. Porém, de nada adiantam estas propriedades se uma pessoa puder emitir certificados em nome de outras pessoas. É necessário a presença de uma entidade confiável, chamada Autoridade Certificadora (AC), para certificar que as informações presentes num certificado digital pertencem a determinada pessoa.

As ACs normalmente delegam a função de fazer a interação com o requerente do certificado à Autoridades de Registro (ARs). A AR tem o papel de receber o pedido de requisição do requerente do certificado, verificar e validar se os dados presentes na requisição estão de acordo com os dados do requerente. Os dados presentes num certificado variam de acordo com as políticas de cada AC, como exemplo pode ser citado o nome do requerente e seu CPF. Após a validação dos dados, a AR encaminha a requisição à AC que então pode emitir o certificado. Em seguida a AC envia um resposta à AR, contendo o certificado, para que a AR possa encaminhar o certificado ao requerente.

A comunicação entre as partes envolvidas pode ser feita de várias formas, via e-mail por exemplo. Contudo ela envolve sempre os mesmo objetivos, emitir ou revogar certificados digitais. Existem protocolos padronizados que definem as mensagens que necessitam ser trocadas entre AC e AR durante o processo de requisição ou revogação de certificados. Todavia observa-se que muitos *softwares* utilizados para gerenciar ACs e ARs implementam o seu próprio protocolo, causando incompatibilidades com entidades que utilizem outros *softwares*.

O presente trabalho se insere neste contexto, estudar e implementar

um protocolo padronizado para realizar a comunicação entre Autoridade Certificadora e de Registro.

1.1 OBJETIVOS

O objetivo deste trabalho é implementar um protocolo para tornar possível a comunicação, de forma segura, entre Autoridades Certificadoras e Autoridades de Registro em Sistemas Gerenciadores do Ciclo de Vida de Certificados Digitais (SGCs).

1.1.1 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- apresentar e analisar protocolos para a comunicação entre Autoridades Certificadoras e de Registro;
- mapear as estruturas ASN.1 do protocolo para XML;
- implementar uma biblioteca para facilitar a geração e manipulação das mensagens do protocolo;
- analisar e descrever as mudanças necessárias ao Sistema de Gerenciamento de Certificados Digitais ICPEDU (SGCI) para suportar o protocolo implementado;
- integrar o protocolo implementado ao SGCI;

1.2 JUSTIFICATIVA E MOTIVAÇÃO

O Laboratório de Segurança em Computação (LabSEC) da Universidade Federal de Santa Catarina (UFSC), local onde este trabalho foi realizado, tem desenvolvido vários trabalhos na área de certificação digital e suas aplicações. Alguns destes trabalhos relacionam-se com o desenvolvimento de Sistemas Gerenciadores do Ciclo de Vida de Certificados Digitais (SGCs).

Um requisito desejável para este tipo de sistema é a interoperabilidade. Com sistemas interoperáveis é possível, por exemplo, utilizar um sistema com requisitos maiores de segurança para gerenciar Autoridades Certificadoras (ACs), e outro sistema *online* para gerenciar as Autoridades de Registro (ARs), para que os usuários possam requisitar seus certificados através de

uma interface *web*. Entretanto, os diferentes sistemas até hoje desenvolvidos no LabSEC não fazem uso de protocolos padronizados de comunicação entre AC e AR , não sendo possível adotar a abordagem descrita acima.

Através das atividades desenvolvidas pelo autor dentro do LabSEC, surgiu a oportunidade de estudar os protocolos presentes na literatura para a comunicação entre ACs e ARs e implementar um dos protocolos no SGCI.

1.3 METODOLOGIA

Para se alcançar os objetivos deste trabalho foram estudadas normas que descrevem protocolos para fazer a comunicação segura entre Autoridades Certificadoras e de Registro, de forma a garantir a integridade e autenticidade das mensagens trocadas entre as entidades.

As avaliações de cada protocolo estudado foram realizadas analisando a abrangência e a necessidade de cada um deles. Com isto foi proposta a implementação de um dos protocolos estudados para o SGCI.

Após esta análise foi implementada uma biblioteca para facilitar a geração das mensagens do protocolo dentro do SGCI. Em seguida foram estudadas e implementadas as alterações necessárias ao SGCI para suportar tal protocolo. Por fim, foi implementada uma camada de transporte, para o envio das mensagens entre diferentes instâncias do SGCI.

1.4 LIMITAÇÕES DO TRABALHO

No presente trabalho não será abordada a comunicação entre a AR e a entidade final, focando-se apenas na comunicação entre a AR e a AC. E serão abordadas apenas as mensagens que necessitam ser trocadas entre as entidades para a requisição, emissão e revogação de certificados digitais.

Embora existam outros modelo de Infraestruturas de Chaves Públicas e certificados digitais, como o *Pretty Good Privacy* (CALLAS et al., 2007), o escopo deste trabalho restringe-se apenas ao padrão X.509.

1.5 ESTRUTURA DO TRABALHO

No capítulo 2 são apresentados conceitos básicos relacionados à certificação digital, como criptografia simétrica e assimétrica, infraestruturas de chaves públicas e sistemas gerenciadores de certificados.

No capítulo 3 será abordada a implementação do protocolo de geren-

ciamento de certificados (CMP), escolhido para este trabalho, e como ele é utilizado neste trabalho.

No capítulo 4 são apresentadas as alterações feitas no código fonte do SGCI para a integração com o protocolo CMP e a camada implementada no SGCI para fazer o transporte das mensagens do CMP entre diferentes instâncias do SGCI.

Por fim, no capítulo 5, as considerações finais obtidas através da pesquisa deste trabalho e sugestões de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 CRIPTOGRAFIA

Criptografia, do Grego kryptós (oculto) e gráphein (escrita), é a ciência que estuda as técnicas de transformação de uma informação de forma a deixá-la ilegível. O primeiro uso que se tem registro data de 1900 a.c., no Egito, contudo uma das técnicas mais clássicas da criptografia, a cifra de César, foi utilizada por Júlio César, na antiga Roma, em torno de 50 a.c.

Atualmente, a criptografia está presente no cotidiano da maioria das pessoas, sendo amplamente utilizada na Web, transações financeiras e diversas outras aplicações. Por exemplo, toda vez que efetuamos uma operação bancária, utilizando o *internet banking* ou até mesmo o caixa eletrônico, estamos fazendo uso de criptografia.

O processo de cifragem e decifragem de uma mensagem depende de um algoritmo. Um algoritmo criptográfico é uma transformação matemática que pode funcionar de duas maneiras distintas: uma para o processo de cifragem e outra para a decifragem de uma mensagem. Os algoritmos necessitam de duas informações: a própria mensagem que será cifrada/decifrada e uma chave, que servirá como parâmetro para as transformações matemáticas sobre a mensagem. Uma chave é uma cadeia aleatória de bits. Para cada par de mensagem e chave, um mesmo algoritmo deve gerar resultados distintos.

Os algoritmos criptográficos devem satisfazer pelo menos um dos seguintes aspectos relacionados à segurança da informação (SCHNEIER, 1996):

- **confidencialidade:** A mensagem deve ser legível apenas por quem estiver autorizado;
- **autenticidade:** O destinatário da mensagem deve ser capaz de identificar sua origem;
- **integridade:** O destinatário da mensagem deve ser capaz de verificar se a mensagem foi modificada. Um atacante não deve ser capaz de substituir uma mensagem falsa por uma legítima;
- **não-repúdio** ou **irretratabilidade:** Um emissor não deve ser capaz de negar a autoria de uma mensagem produzida por ele.

2.1.1 Resumo criptográfico

Funções de resumo criptográfico, também conhecidas como funções de *hash*, são funções matemáticas que, para cada valor de entrada, produzem uma saída diferente de tamanho fixo. Esta saída é comumente chamada de impressão digital, pois identifica unicamente a mensagem de entrada(SCHNEIER, 1996). A figura 1 exemplifica o funcionamento das fun-

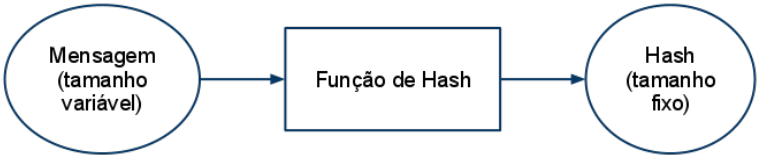


Figura 1: Função de Hash

ções de hash.

Uma função de resumo criptográfico F deve satisfazer as seguintes propriedades(STALLINGS, 2005) :

- F pode ser aplicada em mensagens de qualquer tamanho;
- F produz uma saída de tamanho fixo;
- o cálculo do hash é relativamente fácil de se calcular;
- é computacionalmente inviável recriar uma mensagem a partir do seu hash;
- é computacionalmente inviável produzir duas mensagens distintas que, quando aplicadas a F, resultem no mesmo hash.

Atualmente um dos algoritmos de resumo criptográfico mais utilizado é o *Secure Hash Algorithm 1* (SHA-1). Porém o *National Institute of Standards and Technology* (NIST), recomenda que a partir de janeiro 2011 seja abandonado o uso do SHA-1 (BARKER; ROGINSKY, 2011). Como alternativa, existe a família de algoritmos SHA-2.

2.1.2 Criptografia Simétrica

Na criptografia simétrica, a mesma chave deve ser usada pelo emissor, para cifrar a mensagem, e pelo destinatário, para decifrar a mensagem.

Na figura 2, Alice deseja trocar uma mensagem particular com Bob.

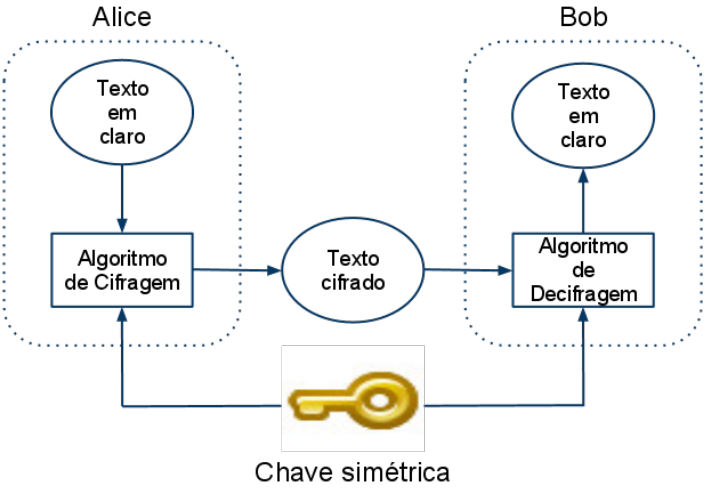


Figura 2: Criptografia Simétrica

Alice aplica um algoritmo de cifragem na mensagem, fornecendo uma chave e envia para Bob a saída do algoritmo, que é a mensagem cifrada. Para Bob obter a mensagem original, ele aplica um algoritmo de decifragem na mensagem cifrada, fornecendo a mesma chave utilizada por Alice na cifragem e obtém a mensagem em claro.

A criptografia simétrica é uma abordagem simples, muito utilizada para cifrar grandes quantidades de dados, devido ao seu alto desempenho computacional. Apesar de sua simplicidade, existem alguns pontos críticos desta abordagem que devem ser considerados. A gerência das chaves pode se tornar muito custosa para uma rede com muitos usuários, pois cada entidade necessita de uma chave diferente para se comunicar de forma segura com cada uma das outras entidades. Por exemplo, em uma rede com x entidades, existiram $\frac{x(x-1)}{2}$ chaves. Outro fator importante é o estabelecimento da chave entre duas entidades. É necessário que as entidades estabeleçam a chave através de um canal seguro, pois caso a chave seja interceptada por um atacante, este poderá decifrar todas as mensagens trocadas entre as entidades que utilizarem esta chave.

Existem diversos algoritmos de criptografia simétrica em uso atualmente, entre eles vale citar o *Data Encryption Standard* (DES) (NIST, 1977)

e o seu sucessor, o *Advanced Encryption Standard* (AES) (NIST, 2001).

2.1.3 Criptografia Assimétrica

Em 1976, os pesquisadores Whitfield Diffie e Martin Hellman propuseram, na publicação do artigo *New Directions in Cryptography* (DIFFIE; HELLMAN, 1976), uma alternativa para resolver o problema da troca de chaves entre emissor e receptor. Este artigo deu origem à criptografia assimétrica, onde ao invés de uma única chave, é usado um par de chaves, sendo uma chave pública (conhecida por todos) e outra privada (secreta). Quando geradas sob certas propriedades matemáticas, essas chaves permitem que mensagens cifradas com a chave pública só possam ser decifradas com a chave privada e vice-versa.

Para Alice trocar uma mensagem secreta com Bob, ele deve obter a chave pública de Bob e então aplicar um algoritmo de cifragem na mensagem, fornecendo a chave pública de Bob. O resultado é a mensagem cifrada. Para Bob conseguir ler o conteúdo da mensagem, ele deve aplicar um algoritmo de decifragem na mensagem cifrada, fornecendo sua chave privada. A figura 3 esquematiza o funcionamento da criptografia assimétrica.

A criptografia assimétrica resolve os problemas da criptografia simétrica de troca de chaves. Uma rede com x entidades necessita de apenas x pares de chaves, ou seja $2x$ chaves. Também elimina a necessidade de um canal seguro para a troca de chaves, já que a chave que deve ser distribuída é pública. A criptografia assimétrica apresenta uma desvantagem em relação a criptografia simétrica, o desempenho computacional (SCHNEIER, 1996). O tempo necessário para cifrar dados utilizando criptografia assimétrica é maior se comparado com a criptografia simétrica. Devido a este motivo, é muito comum o uso em conjunto da criptografia simétrica e assimétrica.

O PKCS#7 (KALISKI, 1998) , quando utilizado para armazenar dados cifrados, é um exemplo do uso em conjunto da criptografia simétrica e assimétrica. Primeiro uma chave simétrica é gerada aleatoriamente. A mensagem é cifrada utilizando a chave simétrica gerada e esta chave é então cifrada com uma chave pública. Para obter a mensagem original utiliza-se a chave privada, correspondente à chave pública utilizada na cifragem, para decifrar a chave simétrica. Esta é usada para decifrar a mensagem propriamente dita, como ilustrado na figura 4.

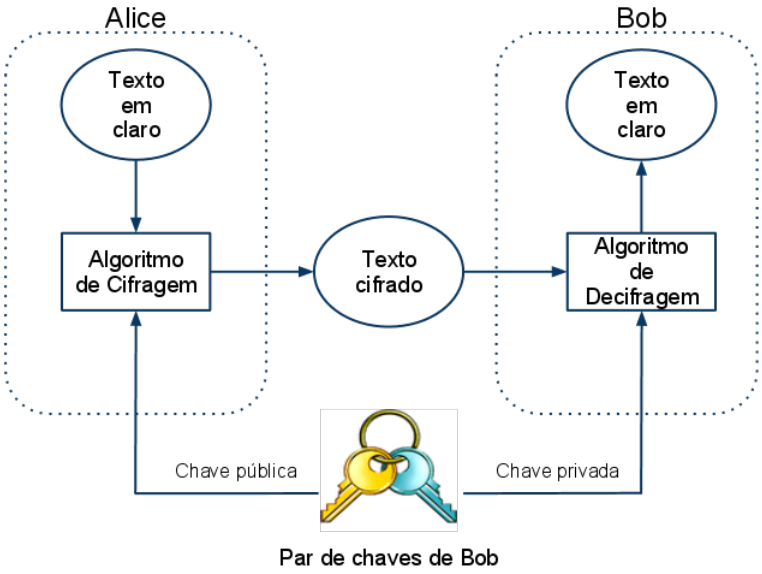


Figura 3: Criptografia Assimétrica

2.1.4 Assinatura digital

A assinatura digital é uma sequência de bits que atestam que o assinante conhece o conteúdo do documento eletrônico assinado. O processo de assinatura consiste em cifrar o hash da mensagem com a chave privada do assinante. O documento assinado é a mensagem mais a sua assinatura. Para verificar a assinatura de um documento, ela é decifrada com a chave pública do assinante. Este valor é comparado com o hash presente no documento. A igualdade destes valores indica que o documento está íntegro e é autêntico. A figura 5 esquematiza o funcionamento da assinatura digital.

Algoritmos que implementam assinatura digital devem satisfazer as seguintes propriedades (STALLINGS, 2005):

- a assinatura deve ser uma sequência de bits que depende da mensagem que está sendo assinada;
- a assinatura deve utilizar alguma informação única do assinante para

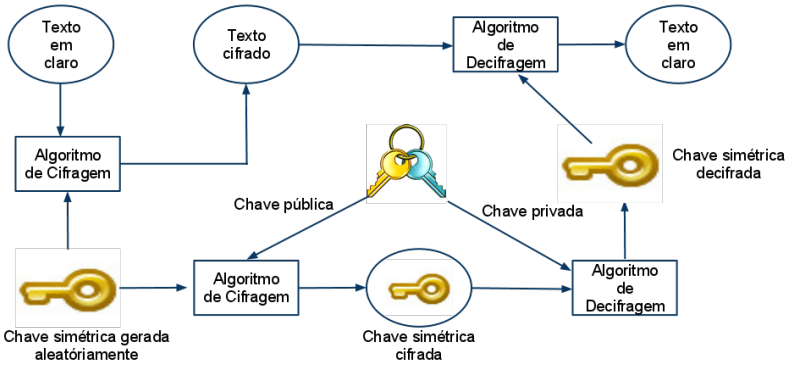


Figura 4: Uso de criptografia assimétrica em conjunto com criptografia simétrica

evitar a falsificação e negação;

- a assinatura de um documento deve ser um processo simples;
- a verificação de uma assinatura deve ser um processo simples;
- deve ser computacionalmente inviável forjar uma assinatura;
- deve ser prático preservar uma cópia da assinatura em algum repositório.

Atualmente o algoritmo mais utilizado para assinatura digital é o criado por Rivest, Shamir e Adleman (RSA) (JONSSON; KALISKI, 2003), mas vale citar também o *Elliptic Curve Digital Signature Algorithm* (ECDSA) (JOHNSON; MENEZES; VANSTONE, 2001).

2.2 INFRAESTRUTURA DE CHAVES PÚBLICAS

Uma Infraestrutura de Chaves Públicas (ICP) consiste de um conjunto de técnicas, práticas e procedimentos que visa dar suporte a aplicações de criptografia de chave pública. Uma ICP baseia-se em um sistema de confiança mútua, no qual duas entidades confiam em uma terceira, para que possam verificar e confirmar a identidade de ambas as partes. Esta terceira parte é usualmente chamada de âncora de confiança.

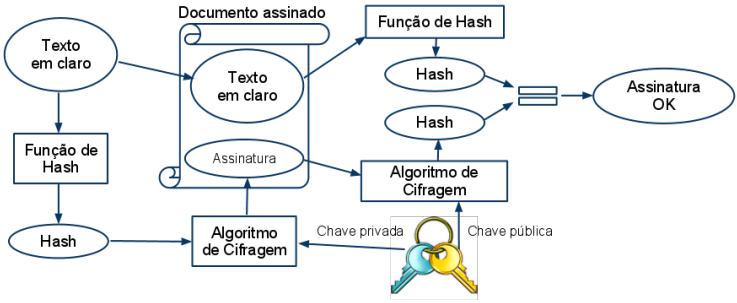


Figura 5: Assinatura digital

2.2.1 Certificado digital

Certificado digital (KONFELDER, 1978) é um documento eletrônico que contém uma chave pública e informações sobre um indivíduo, detentor da chave pública. Outras informações que comumente aparecem em um certificado digital são o número serial do certificado, período de validade, nome e assinatura do emissor do certificado (HOUSLEY; POLK, 2001).

O padrão de certificados digitais mais utilizado atualmente é o X.509v3 (COOPER et al., 2008). Além das informações comuns a todo certificado digital mencionadas anteriormente, certificados neste padrão podem conter outros campos, opcionais, como extensões, que os tornam mais flexível para serem usados por diferentes aplicações. Como exemplo, pode-se citar a extensão *Key usage* que define qual o uso da chave privada do certificado.

2.2.2 Lista de certificados revogados

A Lista de Certificados Revogados (LCR) é uma lista contendo os números seriais de certificados não expirados que não são mais confiáveis. A responsabilidade pela geração e publicação desta lista é do emissor de certificado.

Um certificado pode ser inserido em uma LCR por diversos motivos. O emissor do certificado pode tomar a decisão de colocá-lo na LCR, dependendo de suas políticas, mas também o dono do certificado pode solicitar a revogação de seu certificado, por exemplo, se a sua chave privada for comprometida.

Os principais campos presentes em uma LCR são (COOPER et al., 2008):

- o emissor da LCR;
- a data de emissão da LCR;
- a data da próxima emissão;
- uma lista dos certificados revogados, contendo o número serial e a data de revogação de cada certificado;
- assinatura do emissor da LCR, dando autenticidade ao documento.

2.2.3 Autoridade Certificadora

A Autoridade Certificadora (AC) é a base de toda ICP (HOUSLEY; POLK, 2001). Ela é composta por hardware, software e pessoas que a operam e é a terceira parte confiável para a emissão de um certificado digital.

A AC é conhecida por dois atributos: seu nome e sua chave pública, e realiza as seguinte funções dentro da ICP (HOUSLEY; POLK, 2001):

- emite certificados, ou seja, cria e assina os certificados;
- mantém informações sobre os estados dos certificados e emite LCRs;
- publica os certificados (não expirados) e LCRs;
- mantém arquivos de informação sobre os certificados expirados ou revogados emitidos por ela.

Para poder atender os itens descritos acima, uma AC pode delegar algumas funções a outras entidades (HOUSLEY; POLK, 2001). Dois exemplos comuns onde ocorre a delegação de responsabilidades por uma AC são:

- criação de uma hierarquia, de forma que cada AC seja responsável por apenas um grupo de requerentes de certificado. Por exemplo, uma empresa pode ter uma AC, denominada autoridade certificadora raiz, responsável por emitir certificados apenas para outras entidades. E subjugadas a ela, outras ACs, denominadas autoridades certificadoras intermediárias ou finais, sendo cada uma responsável por apenas um setor da empresa. A figura 6 mostra um exemplo de uma empresa com 3 ACs.

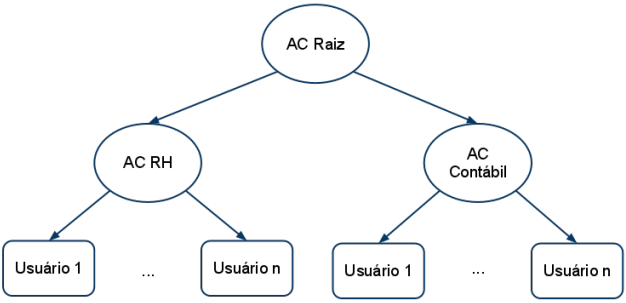


Figura 6: ICP hierárquica

A AC Raiz é responsável apenas por emitir certificados para outras ACs. Enquanto que a AC RH e a AC Contábil são responsáveis para emitir certificados para os funcionários do setor de recursos humanos e contábil, respectivamente;

- delegação da verificação dos requerentes de certificado a uma autoridade de registro.

2.2.4 Autoridade de Registro

A Autoridade de Registro (AR) é responsável por verificar a identidade e outras informações relevantes de usuários que desejam obter um certificado ou que já possuem um certificado e desejam revogá-lo.

Quando uma AC deseja fazer uso de ARs, ela estabelece um relacionamento de confiança com uma ou mais ARs, delegando a função de verificar a validade dos dados submetidos pelas entidades que solicitam um certificado. A entidade que deseja um certificado dessa AC deve apresentar-se a uma das ARs confiadas pela AC, normalmente comparecendo ao local físico onde está instalada a AR, onde o requerente será identificado e seus dados validados para então a AR enviar uma requisição de emissão de certificado à AC, que emite o certificado, validado pela AR confiável.

2.2.5 Modelos de transação

Como visto anteriormente, uma AC pode ou não delegar algumas de suas responsabilidades a uma AR, dependendo de suas políticas. Para certas aplicações, a presença da AR traz uma complexidade desnecessária à aplicação. Por exemplo, ACs especializadas em emitir certificados de correio eletrônico não necessitam validar a identidade do requerente, mas apenas que ele tem acesso ao endereço de correio eletrônico para o qual o certificado será emitido. Esta validação pode ser feita de forma automatizada, por exemplo através do protocolo desafio-resposta (ZORN, 1999). Este tipo de abordagem, onde não há a presença de uma AR, é conhecido como modelo de transação de duas partes.

A figura 7 ilustra como funciona o modelo de duas partes no software AC Correio (VIGIL, 2007).

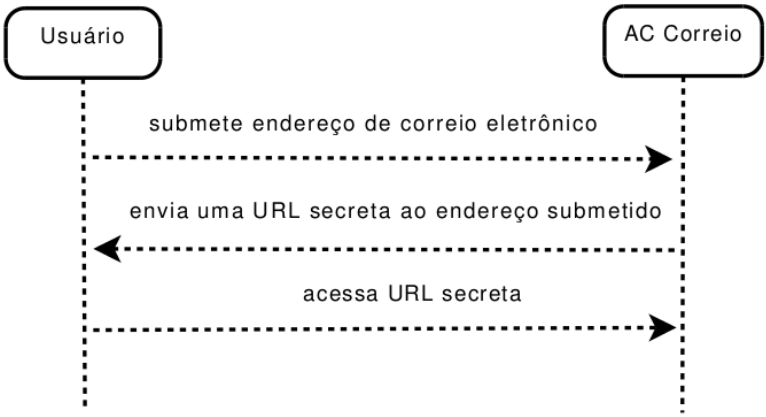


Figura 7: Protocolo de desafio-resposta da AC Correio [(VIGIL, 2007) p.25]

O requerente do certificado informa seu endereço de correio eletrônico à AC Correio, através de seu site. Em seguida a AC Correio envia uma URL secreta ao endereço fornecido pelo usuário, que acessa esta URL. Os próximos passos são a geração do par de chaves do usuário e emissão do certificado.

Contundo, para aplicações que necessitam uma validação mais rígida dos dados do requerente do certificado, é recomendável o uso do modelo de

transação de três partes. A principal responsabilidade de uma AC é proteger sua chave privada, por conta disto ACs normalmente estão localizadas em salas cofre, ambientes com acesso controlado, o que pode dificultar a AC de interagir diretamente com o usuário final (HOUSLEY; POLK, 2001). Nesta abordagem insere-se o papel da AR, responsável por verificar a veracidade dos dados fornecidos pelo requerente do certificado.

2.3 SGC

Um Sistema de Gerenciamento de Certificados (SGC) é um software responsável pela gestão do ciclo de vida de certificados digitais. Alguns dos principais requisitos são:

- criação e operação de autoridades certificadoras;
- emissão e revogação de certificados digitais;
- emissão da lista de certificados revogados (LCR).

2.3.1 SGCI

O Sistema de Gerenciamento de Certificados Digitais da Infraestrutura de Chaves Públicas para Ensino e Pesquisa (SGCI) é um SGC desenvolvido para o âmbito acadêmico, fazendo parte do projeto ICPEDU, financiado pela Rede Nacional de Ensino e Pesquisa (RNP). Ele foi concebido para utilizar somente software livre, ser flexível, simples e robusto, provendo as funcionalidades necessárias para o gerenciamento de ICPs.

Este software é divido em cinco módulos, descrito a seguir:

- Criador: Responsável pelas configurações do software e disponibilização de entidades e usuários. Neste módulo é possível cadastrar e deletar ACs, ARs e usuários, além de vincular usuários com as entidades;
- Administrador de AC: Responsável pela administração de autoridades certificadoras. Neste módulo é possível cadastrar e deletar operadores, cadastrar modelos de certificados, gerenciar relacionamentos de confiança e definir o período para emissão de LCRs;
- Administrador de AR: Responsável pela administração de autoridades de registro. Neste módulo é possível cadastrar e deletar operadores e gerenciar relacionamentos de confiança;

- Operador de AC: Responsável pela operação de autoridades certificadoras. Neste módulo é possível aprovar ou rejeitar requisições de certificado e de revogação, emitindo os certificados para as requisições aprovadas, e emitir LCRs;

- Operador de AR: Responsável pela operação de autoridades de registro. Neste módulo é possível aprovar ou rejeitar requisições de certificado e de revogação, encaminhando as requisições aprovadas para a AC responsável.

Atualmente o Laboratório de Segurança em Computação (LabSEC) é responsável pelo desenvolvimento e manutenção do SGCI que possui uma versão estável, 1.3.7, e uma nova versão, 2.0.0, encontra-se em desenvolvimento.

2.4 *CERTIFICATE MANAGEMENT PROTOCOL*

O Certificate Management Protocol (CMP) (ADAMS et al., 2005) é um protocolo utilizado para criação e gerenciamento de certificados digitais X.509v3 (COOPER et al., 2008) e define mensagens que permitem a interação online de diferentes componentes de uma ICP.

A figura 8 mostra como funciona a troca de mensagens entre o usuário final, a AR e a AC durante uma requisição de certificado. O usuário submete a requisição no formato PKCS#10 (TURNER, 2010) à AR. Para garantir que o usuário que enviou a requisição realmente está em posse da chave privada utilizada para assinar a requisição, a AR envia ao usuário um desafio, por exemplo, um número gerado aleatoriamente. O usuário cifra este número com sua chave privada e envia o número cifrado de volta para a AR. A AR então decifra este número com a chave pública presente na requisição enviada anteriormente e compara com o número gerado. Se os dois números forem iguais ela então pode continuar a processar a requisição. O próximo passo é montar a mensagem que será enviada para a AC. A AR encaminha o pedido de requisição à AC que emite o certificado e envia a resposta para a AR. O certificado é então enviado para o usuário.

No exemplo acima, as mensagens trocadas entre a AC e a AR são definidas pelo CMP. As seções seguintes apresentam uma visão geral sobre as mensagens do CMP para emissão e revogação de certificados.

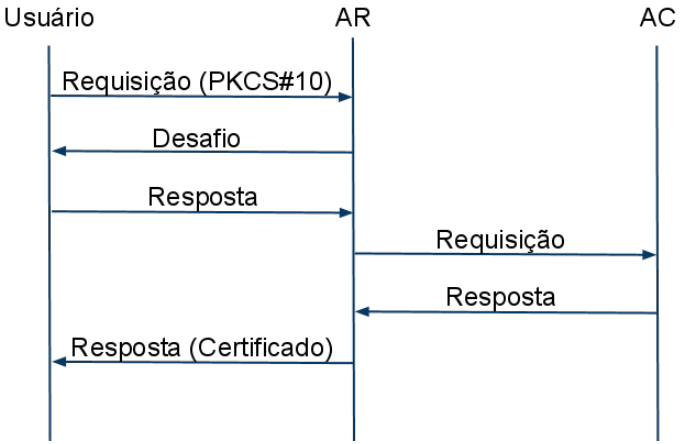


Figura 8: Troca de mensagens entre componentes da ICP segundo o CMP

2.4.1 Mensagem base

Toda mensagem definida pelo CMP pode apresentar os seguintes campos:

- cabeçalho: Apresenta informações comuns a várias mensagens, utilizadas para identificar o emissor e destinatário, por exemplo;
- corpo: Apresenta informações específicas para cada requisição. Este campo será apresentado em detalhes nas seções seguintes;
- proteção: Contém bits que protegem a mensagem. Por exemplo, a assinatura dos campos citados acima. Este campo é opcional;
- certificados extras: Pode ser usado para carregar certificados necessários por uma das partes. Este campo é opcional.

2.4.2 Mensagens para emissão de certificados

Para o processo de requisição e emissão de certificados, existem basicamente duas mensagens, que são trocadas entre a AC e a AR, ilustradas na figura 8 como requisição e resposta.

Na requisição, a AR pode enviar à AC tantas informações sobre a requisição quanto forem necessárias. Podem ser fornecidas informações desde os campos básicos do certificado sendo requisitado, até suas extensões. Porém, a decisão de aceitar ou não cada uma das informações fornecidas pela AR é feita pela AC, na emissão do certificado ou rejeição da requisição. Em ambos os casos a AC envia uma resposta para a AR.

Na resposta da AC, estão presentes o identificador da requisição, obtido na mensagem anterior. O status da requisição, indicando se a requisição foi aprovada, rejeitada ou se está aguardando processamento. Caso seja rejeitada, a AC ainda pode fornecer informações sobre o motivo da rejeição. E por último o certificado emitido. O identificador e o status da requisição são obrigatórios, o certificado é opcional.

2.4.3 Mensagens para revogação de certificados

Para o processo de revogação de certificados, a AR envia à AC um pedido de revogação contendo os seguintes campos:

- detalhes do certificado: A AR pode fornecer tantas informações sobre o certificado quando forem necessárias para a sua identificação. Usualmente o número serial e o campo emissor são suficientes;

- extensões: Caso o pedido de revogação seja aceito pela AC, a AR pode fornecer extensões para a entrada do certificado na LCR. Este campo é opcional.

Após processamento da requisição pela AC, esta deve enviar uma resposta contendo os seguintes campos:

- *status*: O status do pedido de revogação. Este campo é semelhante ao campo status presente na resposta de requisição de certificado, descrito na seção anterior.

- identificação dos certificados: Campo emissor e número serial dos certificados para os quais a revogação foi requisitada. Este campo é opcional.

- LCRs: As LCRs que podem ter sido geradas após o processamento da requisição. Este campo é opcional.

2.4.4 Mensagens para polling

Existem cenários em que a AR precisa pedir o status das requisições enviadas à AC, por exemplo, quando a AC envia uma resposta de requisição de certificado informando que a requisição ainda não foi processada. Nestes cenários a AR envia uma mensagem contendo os identificadores das requisições cujo status ela deseja saber. Dependendo do status destas requisições, a AC pode enviar duas respostas distintas, descritas a seguir:

- se as requisições ainda não foram processadas: neste caso a AC envia uma resposta com os mesmos identificadores da mensagem anterior, enviada pela AR, e o tempo para a AR esperar antes de fazer a próxima solicitação;

- se algumas das requisições tiverem sido processadas: neste caso a AC envia uma resposta para as requisições já processadas. Esta mensagem é a mesma enviada pela AC durante uma requisição de certificado, descrita na seção 2.4.2.

2.4.5 Protocolo para o transporte das mensagens

O CMP necessita de mecanismos de transporte bem definidos para a troca de mensagens entre entidades. Os mecanismos de transporte definidos na primeira versão do CMP (ADAMS; FARRELL, 1999) foram retirados da segunda versão (ADAMS et al., 2005), sendo tratados em um documento separado (KAUSE; PEYLO, 2011). Este documento ainda encontra-se em desenvolvimento, porém apresenta mecanismos bem definidos para o transporte do CMP.

Para a comunicação *online* entre duas entidades e quando um protocolo confiável como o TCP está disponível, o protocolo HTTP deve ser utilizado para o transporte das mensagens do CMP, pois ele fornece, através de seus status, o necessário para reportar erros (KAUSE; PEYLO, 2011). Problemas gerais no servidor ou erros causados por requisições inválidas podem ser facilmente reportadas ao cliente.

O Internet Assigned Numbers Authority (IANA), órgão responsável pela alocação de endereços IP a nível global e outros recursos relacionados à internet, definiu o campo *Content-Type* do cabeçalho HTTP como "*application/pkixcmp*" para as mensagens do CMP. Toda requisição ou resposta HTTP que estiver carregando uma PKIMessage, deve setar este este valor no cabeçalho.

As mensagens do CMP devem ser enviadas no corpo de uma requisição POST do HTTP. O servidor que receber uma requisição, deve enviar uma reposta do CMP no corpo da resposta HTTP, com o status 200. Outros status não devem ser usados neste caso. Para evitar que o cliente obtenha uma resposta em cache, clientes e servidores devem adicionar no cabeçalho HTTP o seguinte: “Cache-Control: no-cache”.

Alternativamente ao HTTP, o protocolo TCP pode ser utilizado, porém seu uso não é recomendado, sendo o HTTP o protocolo indicado para fazer o transporte do CMP (KAUSE; PEYLO, 2011). Por este motivo, este trabalho não abordará o TCP.

Existem alguns aspectos que devem ser considerados, quando este protocolo é usado para trocar mensagens pela internet. Eles são descritos a seguir (KAUSE; PEYLO, 2011):

- existe o risco de ataques de negação de serviço com o envio de várias requisições ao servidor ao mesmo tempo. Por isso conexões inativas devem ser fechadas após um determinado *timeout* ou dependendo dos recursos disponíveis;

- o protocolo HTTP não provê segurança. Por isso aconselha-se que seja usado o HTTP sobre SSL. Além disso, é importante que os servidores verifiquem a integridade e autenticidade das mensagens, através da assinatura das mesmas, antes de executar qualquer processamento que altere o estado da transação.

2.4.6 Bibliotecas criptográficas

Nesta seção serão apresentadas algumas bibliotecas criptográficas que possuem suporte ao CMP.

A cryptlib (Digital Data Security Limited, 2011) é uma biblioteca criptográfica desenvolvida na linguagem C, amplamente utilizada e possui uma implementação bastante completa do CMP. Além do CMP, esta biblioteca possui suporte a SSL, S/MIME, PGP, entre outros serviços.

A cmpForOpenSSL (Martin Peylo, 2011) é uma biblioteca desenvolvida, na linguagem C, por um dos autores do protocolo de transporte do CMP. Ela ainda está em desenvolvimento e possui apenas algumas funcionalidades do CMP implementadas. Ao contrário da cryptlib, esta implementa apenas o CMP. Ela foi projetada para funcionar como uma extensão do OpenSSL (The OpenSSL Project, 2011).

2.5 *CERTIFICATE MANAGEMENT OVER CMS*

O Certificate Management Over CMS (CMC) (SCHAAD; MYERS, 2008a) é um protocolo utilizado para criação e gerenciamento de certificados digitais X.509v3 (COOPER et al., 2008) e define mensagens que permitem a interação online de diferentes componentes de uma ICP.

O CMC utiliza os padrões Cryptographic Message Syntax (CMS) (HOUSLEY, 2009) e PKCS#10 (TURNER, 2010) para a codificação de suas mensagens. Como alternativa ao PKCS#10, o Certificate Request Message Format (CRMF) (SCHAAD, 2005) pode ser usado em conjunto com o CMS. As seções a seguir descrevem as mensagens necessárias para emissão e revogação de certificados e os protocolos disponível para fazer o transporte destas mensagens.

2.5.1 Mensagens para emissão de certificados

A requisição de certificado pode ser tanto um PKCS#10 quanto um PKCS#7. O PKCS#10 é o formato mais comum para requisições de certificado, enquanto que o PKCS#7 pode ser utilizado pela AR, para enviar várias requisições em uma única mensagem para a AC. Se o PKCS#7 for utilizado, a requisição pode ser um PKCS#10, uma requisição de acordo com o CRMF ou outra estrutura definida pela própria ICP.

A resposta para a requisição é um PKCS#7 contendo o certificado emitido e, adicionalmente, os certificados necessários para o usuário montar o caminho de certificação.

A figura 9 mostra como funciona a troca de mensagens entre o usuário final, a AR e a AC. O usuário submete a requisição no formato PKCS#10 à AR, que pode encaminhá-la diretamente para a AC ou, se desejar alterar o conteúdo da requisição ou fornecer informações adicionais à AC, montar uma nova requisição e enviá-la dentro de um PKCS#7 assinado (KALISKI, 1998). A AC então emite o certificado e o retorna para a AR dentro de um PKCS#7 assinado, que encaminha para o usuário.

2.5.2 Mensagens para revogação de certificados

A requisição de revogação contém informações sobre o certificado a ser revogado e sobre o motivo da revogação. Estas informações são adicionadas à um PKCS#7 que é assinado pelo requisitante da revogação.

A resposta para o pedido de revogação informa se o certificado foi ou

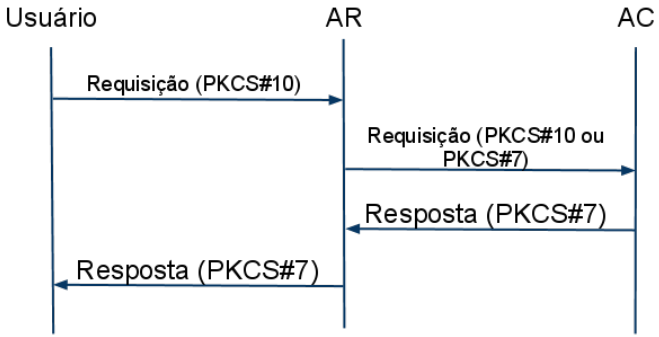


Figura 9: Troca de mensagens entre componentes da ICP segundo o CMC

Tipo da mensagem	Extensão do arquivo
Requisição em PKCS#10	.p10
Requisição em PKCS#7	.crq
Resposta contendo apenas certificados	.p7c
Resposta contendo informações de controle	.crp

Tabela 1: Identificação dos arquivos no CMC. Adaptado de [(SCHAAD; MYERS, 2008b) p.1]

não revogado e, no caso de rejeição, pode conter informações do motivo da rejeição. A resposta também deve ser encapsulada em um PKCS#7.

2.5.3 Protocolos para o transporte das mensagens

Os protocolos de trasporte das mensagens do CMC são definidos na RFC5273 (SCHAAD; MYERS, 2008b). O primeiro deles é baseado em arquivos, normalmente utilizado em sistemas offline. Quando arquivos são utilizados, exitem duas restrições que devem ser observadas. Os arquivos devem conter apenas uma requisição ou resposta presente no arquivo. As extensões do arquivo também variam de acordo com o propósito da mensagem, como pode ser visto na tabela 1.

Tipo da mensagem	Valor do campo content-type
Requisição em PKCS#10	application/pkcs10
Requisição em PKCS#7	application/pkcs7-mime
Resposta contendo apenas certificados	application/pkcs7-mime
Resposta contendo informações de controle	application/pkcs7-mime

Tabela 2: Identificação do campo content-type no CMC sobre HTTP. Adaptado de [(SCHAAD; MYERS, 2008b) p.2]

Para sistemas online, pode ser usado um protocolo sobre o HTTP ou HTTPS. Em ambos os casos, o protocolo deve seguir as seguintes restrições:

- todas as requisições devem ser enviadas através de um POST;
- requisições processadas com sucesso devem conter, na resposta, o código de status 200;
- o campo content-type do cabeçalho do POST HTTP(S) deve estar de acordo com a tabela 2.

2.6 TECNOLOGIAS UTILIZADAS

Nesta seção serão apresentadas as tecnologias utilizadas durante o desenvolvimento deste trabalho.

2.6.1 OpenSSL

O OpenSSL é uma biblioteca de código aberto, gratuita, que implementa os protocolos para as camadas de sockets e de transporte seguros (SSL e TLS, respectivamente), além de funcionalidades criptográficas de uso geral (The OpenSSL Project, 2011).

Esta biblioteca foi escolhida por ser amplamente utilizada, estável e estar num processo contínuo de aprimoramentos, tendo pelo menos uma nova versão lançada a cada ano. O OpenSSL é utilizado indiretamente neste trabalho, a partir de outras bibliotecas que serão apresentadas nas próximas seções.

2.6.2 LibCryptoSec

A LibCryptoSec (LabSEC, 2011a) é uma biblioteca desenvolvida na linguagem C++, orientada a objetos, que abstrai as funcionalidades de criptografia providas pelo OpenSSL.

A API da LibCryptoSec é mais simples e intuitiva, comparada a do OpenSSL, e também possui uma documentação mais abrangente de sua API. Ela funciona como um invólucro para o OpenSSL e foi escolhida por tornar mais simples o uso das funcionalidades criptográficas providas pelo OpenSSL.

2.6.3 PHP5-LibCryptosec

O PHP5-LibCryptoSec (LabSEC, 2011b) é uma extensão PHP, desenvolvida com o objetivo de disponibilizar funcionalidades criptográficas para o PHP. Este atua como um invólucro da biblioteca LibCryptoSec e pode ser dividido em duas partes:

- biblioteca em C++: A extensão PHP propriamente dita;
- biblioteca em PHP: Uma "camada"sobre a extensão, desenvolvida sob os padrões de orientação a objetos, para tornar mais simples o uso da biblioteca.

A figura 10 demonstra o empacotamento das bibliotecas. No centro há a biblioteca OpenSSL, sobre ela encontra-se a LibCryptoSec e sobre esta há o módulo PHP5-LibCryptoSec, primeiro a implementação em C++ e acima a implementação em PHP. Por fim há a aplicação que irá fazer uso destas bibliotecas.

2.6.4 PHP

O PHP (The PHP Group, 2011) é uma linguagem de programação amplamente utilizada para o desenvolvimento de aplicações Web, com integração à linguagem de marcação HTML. Vale também citar a ótima documentação que o PHP possui.

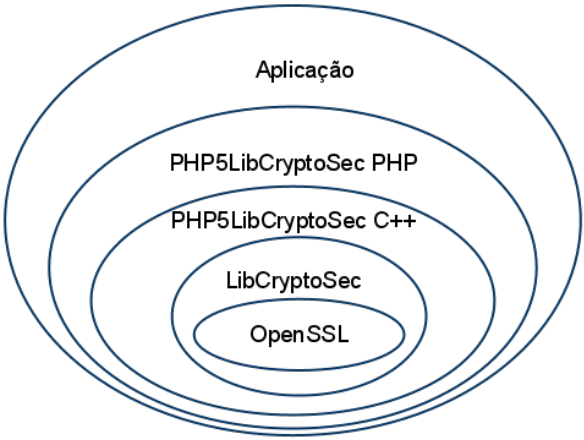


Figura 10: Camadas do módulo PHP5-LibCryptoSec

2.6.5 PHPUnit

O PHPUnit (Sebastian Bergmann, 2011) é um framework de código aberto, que auxilia na criação de testes unitários automatizados para aplicações desenvolvidas em PHP. Outra característica do PHPUnit é que ele possui um utilitário que auxilia na execução e análise dos testes. Além disso possui uma API intuitiva e bem documentada.

2.6.6 XDebug

O XDebug (Derick Rethans, 2011) é uma extensão para PHP que ajuda na depuração de aplicações PHP. A depuração utilizando XDebug pode fornecer as seguintes informações:

- *stack traces* e *function traces* em mensagem de erro com:
  - exibição de todos os parâmetros para funções definidas pelo usuário;

  - nome da função, nome do arquivo e indicações da linha;
  - suporte à classes.

- alocação de memória;
- proteções contra recursões infinitas.

O XDebug também pode ser integrado com o PHPUnit, para gerar métricas como o *code coverage*, e outras ferramentas para tornar a depuração interativa.

2.6.7 Zend Framework

O Zend Framework (Zend Technologies, 2011) é uma biblioteca escrita em PHP, de código aberto e de alta qualidade para o desenvolvimento de aplicações Web. Possui uma API muito completa e bem documentada, com suporte a banco de dados, internacionalização, web services, entre outras várias funcionalidades.

2.6.8 Subversion

O Subversion (SVN) (CollabNet, Inc., 2011) é um sistema de controle de versão de código aberto. Ele permite que várias pessoas trabalhem no mesmo software, editando arquivos diferentes, ou o mesmo arquivo, controlando as edições feitas em cada arquivo e mesclando as diferentes versões do arquivo numa única versão.

Esta ferramenta foi escolhida por facilitar o controle das versões do trabalho e por manter uma cópia atualizada do trabalho em um servidor, como um backup, podendo também obter a última cópia do trabalho em qualquer computador com acesso a internet.

2.6.9 ASN.1

A Abstract Syntax Notation One (ASN.1) (ITU-T, 2011) é uma notação formal para representação, codificação e transmissão de dados. Através de suas regras de codificação, facilita a troca de informações entre aplicações por fornecer uma representação independente de linguagem de programação ou de máquina.

2.6.10 XML

A Extensible Markup Language (XML) (W3C, 2008) é um formato simples de texto para representar informações estruturadas. É um dos formatos mais utilizados atualmente para compartilhamento de informações, entre pessoas ou entre computadores, localmente ou através da internet. O exemplo abaixo mostra como é a sintaxe de um documento XML [1].

```
<?xml version="1.0"?>
<note>
        <to>Tove</to>
        <from>Jani</from>
        <heading>Reminder</heading>
        <body>Don't forget me this weekend!</body>
</note>
```

Código Fonte 2.1: Exemplo de um documento XML

[1] http://www.w3schools.com/XML/note.xml

3 PROTOCOLO DE GERENCIAMENTO DE CERTIFICADOS

Toda comunicação entre AC e AR consiste de, pelo menos, um par de mensagens, que podem ter diversos propósitos. Por exemplo requisição de certificado, requisição de revogação de certificado, etc. Para que diferentes autoridades possam se comunicar, estas mensagens devem ser padronizadas. A seção seguinte faz a análise dos principais protocolos encontrados na literatura para fazer a comunicação entre AC e AR.

3.1 ESCOLHA DO PROTOCOLO

Foram estudados os protocolos CMP (ADAMS et al., 2005) e CMC (SCHAAD; MYERS, 2008a). Um dos fatores analisado é a abrangência dos protocolos, isto é, a variedade de operações que podem ser realizadas dentro de uma ICP suportadas por cada protocolo. Ambos possuem suporte as operações mais básicas, fundamentais a toda ICP, que são a requisição, emissão e revogação de certificados digitais e disponibilização de certificados e lista de certificados revogados. Contudo, o CMP leva vantagem em relação ao CMC, pois ele possui mecanismos para a atualização do par de chaves de Autoridades Certificadoras Raiz e emissão de certificados para certificação cruzada, não presentes no CMC. Apesar deste trabalho não abordar estas operações, considera-se importante que o protocolo as suporte, pois são operações importantes dentro de uma ICP e pode haver necessidade de implementá-las futuramente.

Outro fator analisado é a dependência dos protocolos em relação a outras tecnologias. O CMC utiliza o PKCS#7 (KALISKI, 1998) para encapsular suas mensagens e possibilita o uso do PKCS#10 (TURNER, 2010) para representar as requisições de certificado. Apesar de a utilização destes padrões estar dentro da proposta do CMC, considerou-se esta dependência como uma desvantagem pela necessidade da implementação dos mesmos, caso estes padrões não estejam disponíveis para o ambiente em que o CMC será implementado. No CMC o PKCS#10 pode ser substituído pelo CRMF (SCHAAD, 2005), também utilizado pelo CMP, contudo ainda há a dependência do PKCS#7, não existente no CMP.

A tabela 3 ilustra os fatores analisados acima e apresenta ainda algumas diferenças secundárias entre os protocolos, como por exemplo a porta TCP utilizada para a comunicação entre as entidades. O uso da porta 829 é reservada para o CMP, enquanto que o CMC não possui uma porta padronizada.

Característica	CMP	CMC
Variedade de operações suportadas	Alta	Média
Dependência de outros padrões	Não	Sim
Porta TCP	829	Não padronizada
HTTP Content-type	application/pkixcmp	Depende da mensagem
Transporte das mensagens	Codificação DER das estruturas ASN.1	PKCS#10 ou PKCS#7

Tabela 3: Comparação dos protocolos CMP e CMC

Além dos protocolos citados acima, há o protocolo *Simple Certificate Enrollment Protocol* (SCEP) (PRITIKIN; NOURSE; VILHUBER, 2011), que é semelhante ao CMC, porém não foi considerado neste trabalho por ainda estar em estágio de desenvolvimento.

Após as análises, concluiu-se que o CMP é o protocolo mais adequado para este trabalho, pois possui uma variedade maior de operações que podem ser realizadas com ele, além de não depender de implementações externas. O CMC mostrou-se mais adequado para um sistema onde seja feita a comunicação com o usuário final, fazendo a geração do par de chaves do usuário direto pelo navegador do usuário, pois o CMC utiliza padrões suportados pelos principais navegadores.

3.2 MAPEAMENTO DAS MENSAGENS ASN.1 PARA XML

Para a representação das mensagens definidas pelo CMP optou-se por utilizar o formato XML, por ser amplamente utilizado atualmente para o compartilhamento de informações, além de ser de código aberto e independente de plataforma. A conversão das estruturas ASN.1 descritas na RFC4210 para XML foram feitas utilizando as regras de codificação XML Enconding Rules (XER) (ITU-T, 2001).

As seções a seguir apresentam as mensagens que foram utilizadas em XML e uma descrição sobre cada um de seus campos. As estruturas originais em ASN.1 podem ser encontradas no anexo A.

3.2.1 PKIMessage

PKIMessage é a estrutura básica das mensagens trocadas entre a AC e a AR. Ela contém quatro campos, como pode ser visto no trecho de código 3.1.

O primeiro campo é o cabeçalho da mensagem, descrito pela estrutura PKIHeader, que contém informações para identificar o emissor e destinatário da mensagem. O PKIHeader contém os mesmos campos para todas as mensagens.

O segundo campo é o corpo da mensagem, que fornece as informações do propósito da mensagem. Este campo varia de acordo com o tipo de mensagem enviada.

O terceiro campo é utilizado para proteger a mensagem, podendo, por exemplo, garantir a integridade e autenticidade da mensagem através da assinatura digital.

O quarto e último campo é utilizado para transportar certificados adicionais que possam ser necessários durante uma transação. O terceiro e quarto campo são opcionais.

```
<PKIMessage>
  <header></header> <!-- PKIHeader -->
  <body></body>     <!-- PKIBody -->
  <protection></protection> <!-- PKIProtection, Opcional -->
  <extraCerts></extraCerts> <!-- Opcional -->
</PKIMessage>
```

Código Fonte 3.1: Estrutura PKIMessage em XML

Estes campos serão detalhados nas seções a seguir.

3.2.2 PKIHeader

O PKIHeader contém as informações necessárias para identificar o emissor da mensagem e o destinatário. Além disso pode conter informações adicionais para prover mais informações sobre a mensagem. O trecho de código 3.2 apresenta os campos que constituem o PKIHeader.

```
<PKIHeader>
  <pvno>2</pvno>
  <sender></sender>       <!-- GeneralName -->
  <recipient></recipient> <!-- GeneralName -->
  <messageTime></messageTime>
  <protectionAlg></protectionAlg>
  <senderKID></senderKID>
```

```
  <recipKID></recipKID>
  <transactionID></transactionID>
  <senderNonce></senderNonce>
  <recipNonce></recipNonce>
  <freeText></freeText>
  <generalInfo></generalInfo>
</PKIHeader>
```

Código Fonte 3.2: Estrutura PKIHeader em XML

O campo pvno indica a versão da especificação das estruturas das mensagens do CMP. Nesta implementação este valor será sempre 2, conforme a RFC4210 (ADAMS et al., 2005).

Os campos sender e recipient identificam, respectivamente, o emissor e o destinatário da mensagem. Estes campos importam a estrutura GeneralName da RFC5280 (COOPER et al., 2008). Nesta implementação estes campos conterão o campo subject dos certificados de emissor e destinatário da mensagem.

O campo messageTime contém o horário em que a mensagem foi produzida.

O campo protectionAlg indica o algoritmo utilizado para proteger a mensagem, através do campo protection em PKIMessage. Este campo estará presente sempre que o campo protection estiver presente e ausente caso contrário.

Os campos senderKID e recipKID podem ser utilizados para complementar a identificação de emissor e destinatário, junto dos campos sender e recipient. Eles são identificadores da chave pública de emissor e destinatário, respectivamente. Como não há restrições no SGCI em relação à existência de duas entidades com o mesmo DN no software, estes campos serão usados para uma identificação mais precisa das entidades. Por exemplo, se o software recebe uma mensagem e consta que existem duas ACs com o mesmo DN, o campo recipKID será utilizado para identificar unicamente o destinatário da mensagem.

O campo transactionID é usado para identificar transações. Isto é, mensagens distintas trocadas entre AC e AR que são correlacionadas. Por exemplo uma mensagem de requisição de certificado enviada pela AR poderá gerar duas respostas. A primeira confirmando o recebimento da requisição e informando que a mesma encontra-se em processamento. E uma segunda resposta, após a requisição ser processada, indicando se o certificado foi emitido ou se a requisição foi rejeitada. Este campo irá conter um valor pseudo-aleatório de 128 bits, recomendado pela RFC4210 (ADAMS et al., 2005).

Os campos senderNonce e recipNonce são utilizados para proteger as mensagens e os servidores contra ataques de repetição. O emissor da mensagem irá popular o campo senderNonce com um valor pseudo-aleatório de

128 bits, enquanto que o recipNonce será copiado no senderNonce da mensagem anterior. Caso a mensagem seja a primeira de uma transação o campo recipNonce permanecerá em branco.

O CMP ainda prevê mais dois campos para o PKIHeader, freeText e generalInfo. O primeiro pode ser usado para carregar mensagens para processamento humano enquanto que o segundo pode conter informações adicionais para processamento de máquina.

3.2.3 PKIBody

A estrutura PKIBody é um conjunto de estruturas que identificam o propósito da mensagem. Nas seções seguintes as estruturas necessárias para emissão e revogação de certificados serão apresentadas.

3.2.4 Requisição de certificado

A requisição de certificado contém, no PKIBody, a estrutura CertReqMsg, apresentada no trecho de código 3.3.

```
<CertReqMsg>
  <certReq>
    <certReqId></certReqId>
    <certTemplate>
      <subject></subject>
      <publicKey></publicKey>
    </certTemplate>
  </certReq>
</CertReqMsg>
```

Código Fonte 3.3: Estrutura CertReqMsg em XML

Ela é composta por um identificador, certReqId, e um template de certificado. Este template pode conter tantas informações sobre o certificado quanto o requerente deseje. Esta implementação suporta as seguintes informações:

- *serialNumber*: Representa o número serial do certificado a ser emitido.
- *version*: Representa a versão do certificado a ser emitido. De acordo com a RFC5280 (COOPER et al., 2008), os valores para esta campo podem ser 0, 1 ou 2.
- *subject*: Representa o campo de mesmo nome do certificado a ser emitido.

- *publicKey*: Representa a chave pública do requerente do certificado.

Tanto o certReqId quanto o certTemplate são obrigatórios.

3.2.5 Resposta de requisição de certificado

A resposta de requisição de certificado contém, no PKIBody, a estrutura CertRepMessage, apresentada no trecho de código 3.4.

```
<CertRepMessage>
  <caPubs></caPubs>
  <response>
    <certReqId></certReqId>
    <status>
      <status></status>
      <statusString></statusString>
      <failInfo></failInfo>
    </status>
    <certifiedKeyPair>
      <certOrEncCert>
        <certificate></certificate>
      </certOrEncCert>
    </certifiedKeyPair>
  </response>
</CertRepMessage>
```

Código Fonte 3.4: Estrutura CertRepMessage em XML

O campo caPubs pode conter certificados que possam interessar ao requisitante do certificado. Por exemplo o certificado da AC ou até todo o caminho de certificação. Este campo é opcional.

O campo response, contém o ID da requisição, proveniente da mensagem anterior, o status da requisição e, dependendo do status, o certificado emitido. Se o status for de aprovação, o campo certifiedKeyPair conterá o certificado emitido. Como o certificado é algo público, ele será transportado na mensagem de forma clara, apesar da possibilidade de transportá-lo cifrado. Se o status for de rejeição, ele poderá conter informações sobre o motivo da rejeição, através do campo failInfo. Os diferentes valores deste campo são descritos na estrutura PKIFailureInfo (ADAMS et al., 2005). O status ainda poderá conter uma mensagem para processamento humano, que poderá ser informada durante a geração da mensagem.

O CMP define apenas os campos certReqId e status como obrigatórios, sendo os campos statusString e failInfo opcionais para o status. Nesta implementação, além destes campos, o certificado emitido também estará sempre presente. O campo failInfo estará presente sempre no caso de falha e o campo statusString poderá ser fornecido, durante a geração da mensagem.

3.2.6 Requisição de revogação certificado

A requisição de revogação de certificado contém, no PKIBody, a estrutura RevReqContent, apresentada no trecho de código 3.5.

```
<RevReqContent>
  <RevDetails>
    <certDetails></certDetails> <!-- CertTemplate -->
    <crlEntryDetails></crlEntryDetails> <!-- Extensions -->
  </RevDetails>
</RevReqContent>
```

Código Fonte 3.5: Estrutura RevReqContent em XML

Cada pedido de revogação contém os campos certDetails e crlEntryDetails. O primeiro importa a estrutura CertTemplate (SCHAAD, 2005), onde é possível fornecer tantas informações sobre o certificado a ser revogado quanto forem necessárias. O CMP recomenda que quando o número serial do certificado for conhecido pelo requisitante da revogação, este seja fornecido. O segundo campo, opcional, pode ser utilizado para que o requisitante da revogação forneça as extensões que constem na LCR, caso o certificado seja revogado.

3.2.7 Resposta de requisição de revogação de certificado

A resposta da requisição de revogação de certificado contém, no PKIBody, a estrutura RevRepContent, apresentada no trecho de código 3.6.

```
<RevRepContent>
  <status></status>
  <revCerts>
    <CertId>
      <issuer></issuer>
      <serialNumber></serialNumber>
    </CertId>
  </revCerts>
  <crls></crls>
</RevRepContent>
```

Código Fonte 3.6: Estrutura RevRepContent em XML

A resposta para pedido de revogação contém os campos status, revCerts e crls. O primeiro campo, obrigatório, utiliza a mesma estrutura para o status da resposta de requisição de certificado, descrito na seção 3.2.5. O segundo campo, opcional, pode ser utilizado para informar qual o certificado que foi pedido revogação, através da estrutura CertId, que contém o emissor

e o número serial do certificado. Este campo pode ser usado para aumentar a integridade da mensagem, não sendo necessário para a AR verificar a mensagem anterior para identificar de qual certificado se trata a resposta. O terceiro campo, opcional, pode ser utilizado para fornecer a(s) LCR(s) que possa(m) ter sido emitida(s) após a revogação do certificado.

3.2.8 Pollings

O polling pode ser usado quando a AR deseja saber o status de uma requisição de certificado enviada à AC pendente de resposta, isto é, a AR recebeu a resposta com o status *WAITING*, indicando que a requisição ainda não foi processada pela AC. Para isto, exitem duas mensagens que podem utilizadas, que contém, no campo PKIBody, as estruturas denominadas PollReqContent e PollRepContent.

A estrutura PollReqContent contém apenas o identificador da requisição que deseja-se saber o status, apresentado no trecho de código 3.7.

```
<PollReqContent>
  <certReqId></certReqId>
</PollReqContent>
```

Código Fonte 3.7: Estrutura PollReqContent em XML

A estrutura PollRepContent contém o identificador da requisição presente na mensagem anterior e o tempo para a AR solicitar o próximo polling. Nesta implementação o checkAfter não terá valor semântico, pois a aprovação ou rejeição das requisições pelas ACs de resposta manual, é feita pelo operador da AC, podendo levar uma hora ou um dia, sendo muito difícil estabelecer um tempo para adicionar neste campo. No caso de ACs de resposta automática o certificado é emitido imediatamente e a resposta para a requisição do certificado já terá o certificado emitido. A estrutura PollRepContent é apresentado no trecho de código 3.8.

```
<PollRepContent>
  <certReqId></certReqId>
  <checkAfter></checkAfter>
</PollRepContent>
```

Código Fonte 3.8: Estrutura PollRepContent em XML

Para um polling enviado pela AR, existem duas possibilidades de resposta pela AC, descritas a seguir:

- Caso o certificado ainda não tenha sido emitido, a resposta enviada pela AC conterá, no corpo da mensagem, a estrutura PollRepContent.

- Caso a requisição tenha sido aprovada ou rejeitada pela AC, a resposta conterá, no corpo da mensagem, a estrutura CertRepMessage.

A figura 11 mostra um exemplo envolvendo os dois cenários descritos acima.

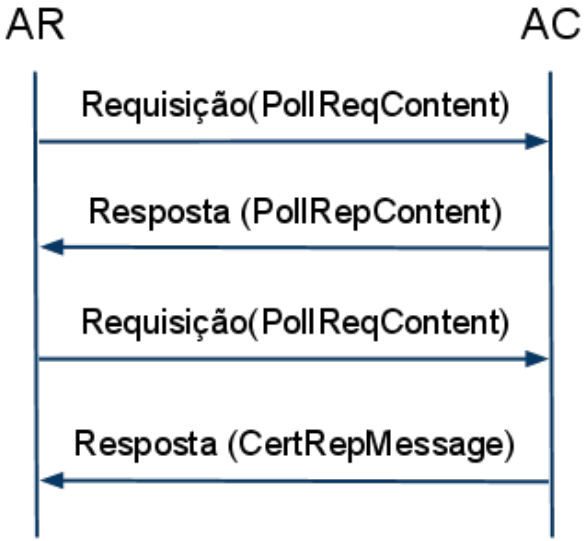


Figura 11: Exemplo do funcionamento de pollings

3.2.9 PKIProtection

As mensagens podem ser protegidas para garantir a integridade. Neste caso, o campo PKIProtection irá conter o valor da assinatura dos campos header e body da mensagem. Adicionalmente, o campo protectionAlg do header deve ser preenchido com o algoritmo utilizado na assinatura.

3.2.10 Validação do XML

Um documento XML por si só não possui informações suficientes para descrever a estrutura que o documento deverá ter. Existem diversas tecnologias que possibilitam definir uma estrutura para documentos XML, as duas principais e recomendadas pela W3C são DTD e XSD. Para este trabalho foi escolhida a XSD, por ser mais expressiva em relação a DTD, simples, e usar a sintaxe do XML.

O trecho de código 3.9 mostra a definição de estrutura PKIMessage em ASN.1. Não existe um documento especificando como fazer a conversão das estruturas ASN.1 para XSD, então a conversão foi feita visando manter a mesma semântica entre as estruturas ASN.1 e XSD.

```
PKIMessage ::= SEQUENCE {
    header              PKIHeader,
    body                PKIBody,
    protection    [0]   PKIProtection OPTIONAL,
    extraCerts    [1]   SEQUENCE SIZE (1..MAX) OF CMPCertificate
                        OPTIONAL
}

PKIHeader ::= SEQUENCE { ... }
PKIBody ::= CHOICE { ... }
PKIProtection ::= BIT STRING
CMPCertificate ::= CHOICE { ... }
```

Código Fonte 3.9: Exemplo de definição da estrutura PKIMessage em ASN.1

Grande parte das definições encontradas em ASN.1 têm um correspondente direto em XSD, porém em alguns casos foi necessário utilizar uma forma alternativa para manter o mesmo efeito. As principais estruturas convertidas que merecem destaque são:

- Estruturas: Estruturas complexas em ASN.1, podem ser declaradas em XSD através da tag *<xs:complexType>*
- Sequências: O elemento SEQUENCE em ASN.1 corresponde à tag *<xs:sequence>*
- Escolhas: O elemento CHOICE em ASN.1 corresponde à tag *<xs:-choice>*
- Campos Opcionais: O atributo OPTIONAL em ASN.1 não tem um correspondente direto em XSD. Para conseguir este efeito, foi utilizado o atributo "minOccurs" com o valor igual a zero, especificando que o elemento pode aparecer zero vezes, ou seja, é opcional.

- Sequências de tamanho indefinido: O atributo SEQUENCE SIZE - (1..MAX) não tem um correspondente direto em XSD. Para este caso, foi utilizado o atributo maxOccurs, com o valor igual a “unbounded”, na tag *<xs:sequence>*. O valor “unbounded” significa que a sequência pode ter tamanho infinito. Para sequências de tamanhos fixos, utiliza-se o atributo “minOccurs” com o valor mínimo e “maxOccurs” com o valor máximo.

Aplicando as regras citadas acima para fazer a conversão, obteve-se o XSD do trecho de código 3.10, correspondente a mesma estrutura definida em ASN.1 no trecho de código 3.9.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="PKIMessage">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="header"/>
                        <xs:element ref="body"/>
            <xs:element ref="protection" minOccurs="0"/>
            <xs:element ref="extraCerts"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="header" substitutionGroup="PKIHeader"/>
<xs:element name="body" substitutionGroup="PKIBody"/>
<xs:element name="protection" substitutionGroup="PKIProtection"/
    >
<xs:element name="extraCerts">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="CMPCertificate"
                             minOccurs="0" maxOccurs="unbounded"
                             />
                </xs:sequence>
        </xs:complexType>
</xs:element>
</xs:schema>
```

Código Fonte 3.10: Exemplo de definição da estrutura PKIMessage em XSD

O XSD de todas as estruturas do CMP utilizadas neste trabalho pode ser encontrado no apêndice A.

3.3 MAPEAMENTO DAS MENSAGENS EM XML PARA OBJETOS

Inicialmente foi feito um levantamento na literatura das bibliotecas já existentes que suportam o CMP. Foram encontradas duas bibliotecas, a *cryptLib* e a *cmpForOpenssl*. Estas bibliotecas estão descritas na seção 2.4.6. Foi desconsiderado o uso destas bibliotecas para este trabalho pelos seguintes motivos:

- são escritas na linguagem C. Desta forma seria ainda necessário portar as funcionalidades para o PHP, de modo a utilizar com o SGCI;
- não possuem suporte a XML.

Não existindo nenhuma biblioteca que satisfaça as necessidades deste trabalho, foi criada uma nova biblioteca, orientada a objetos e em PHP. Um dos objetivos desta biblioteca é fornecer uma interface simples e independente, podendo ser utilizada por diferentes softwares. Além disso, ela é facilmente extensível, possibilitando adicionar as mensagens não tratadas por este trabalho ou fazer implementações customizadas das mensagens.

A estrutura desta biblioteca e suas classes serão descritas nas seções seguintes.

3.3.1 SGCI_PKIMessage

A classe *SGCI_PKIMessage* é responsável pela geração do XML que será transmitido. Ela representa a estrutura PKIMessage descrita na seção 3.2.1 e, semelhante à estrutura, possui três atributos:

- Header: Classe que contém as informações do cabeçalho da mensagem. Esta classe é descrita na seção 3.3.2.
- Body: Classe que contém as informações do corpo da mensagem. Esta classe é descrita na seção 3.3.3.
- Protection: Este atributo contém a assinatura da mensagem. A geração da assinatura da mensagem é descrita na seção 3.3.4.

Além disso, esta classe possui métodos para gerar a mensagem em XML e gerar um objeto PKIMessage a partir de um XML. O método *loadXML* recebe uma string como parâmetro, que deve conter a estrutura PKIMessage em XML, faz o parse deste XML e seta seus atributos de acordo com os valores presentes no XML. Já o método *getXMLEncoded* retorna uma

string contendo o XML da estrutura PKIMessage preenchido com os atributos do objeto. O diagrama apresentado na figura 12 lista todos os métodos

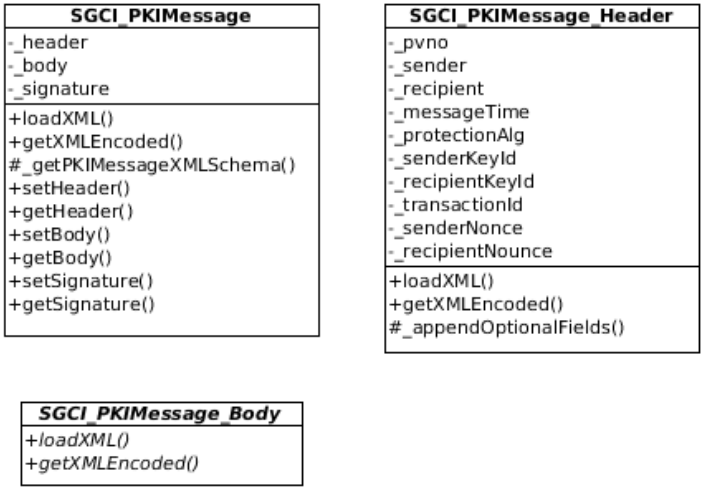


Figura 12: Diagrama de classes da classe PKIMessage

e atributos desta classe, bem como das outras classes com as quais ela se relaciona.

3.3.2 SGCI_PKIHeader

A classe *SGCI_Header* contém as informações do cabeçalho das mensagens e representa o campo *header* da PKIMessage. Ela contém um atributo para cada campo da estrutura PKIHeader descrita na seção 3.2.2. Adicionalmente ela contém métodos para carregar e exportar XMLs. Estes métodos são chamados pela classe *SGCI_PKIMessage*, para carregar/exportar uma mensagem completa.

3.3.3 SGCI_PKIMessage_Body

A classe *SGCI_PKIMessage_Body* corresponde ao campo *body* da PKIMessage. Dependendo do propósito da mensagem, este campo pode conter diferentes informações, como descrito na seção 3.2.3. Para representar estas diferentes informações foram criadas diferentes classes.

A classe *SGCI_PKIMessage_Body* é a superclasse, ela contém os métodos comuns que as outras classes devem implementar, destinados a carregar e exportar XMLs. Toda classe que representar o campo *body* da PKIMessage deve estender esta classe. Algumas das classes implementadas neste trabalho estão ilustradas no diagrama de classes da figura 13. Cada classe possui

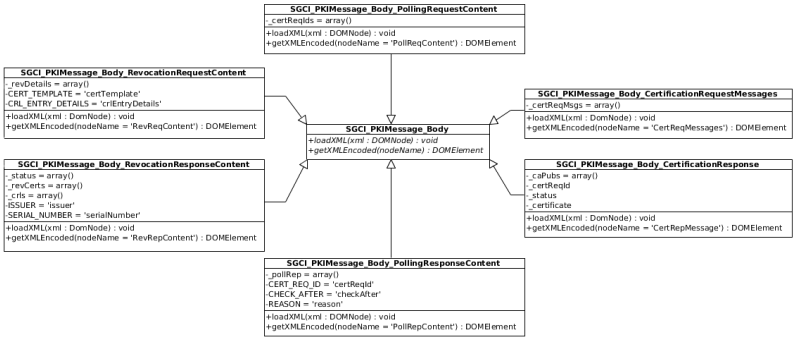


Figura 13: Diagrama de classes da classe Body

métodos para setar e obter os valores de seus atributos, correspondentes aos campos das estruturas referentes de cada classe.

3.3.4 SGCI_PKIMessageSigner

Para garantir a integridade e autenticidade das mensagens trocadas entre as entidades, o CMP prevê um campo de assinatura na estrutura PKIMessage. A classe *SGCI_PKIMessageSigner* é utilizada para gerar e verificar assinaturas para a classe *SGCI_PKIMessage*.

O método *sign* é responsável pela assinatura da PKIMessage. Ele recebe como parâmetros, a PKIMessage que será assinada, o algoritmo de hash e a chave privada que serão usados na assinatura. Opcionalmente é possível

informar a senha da chave privada, caso ela esteja cifrada. A assinatura é gerada a partir dos campos *header* e *body* da PKIMessage. Após gerada a assinatura, este método seta o valor da assinatura na PKIMessage passada como parâmetro e a retorna. Caso algum erro seja gerado durante a assinatura, por exemplo, se o algoritmo de hash informado for inválido, uma exceção é lançada.

A verificação da assinatura de uma PKIMessage é feita através do método *verify*. Este método recebe como parâmetros a PKIMessage assinada e a chave pública para verificar a assinatura. Se a assinatura for válida o método retorna o valor *true*, caso contrário retorna *false*.

Os algoritmos suportados pela classe são os seguintes:

- Para chaves RSA, a assinatura pode ser gerada com os algoritmos de hash sha1, sha224, sha256, sha384 e sha512.

- Para chaves ECDSA, a assinatura pode ser gerada com os algoritmos de hash sha224, sha256, sha384 e sha512.

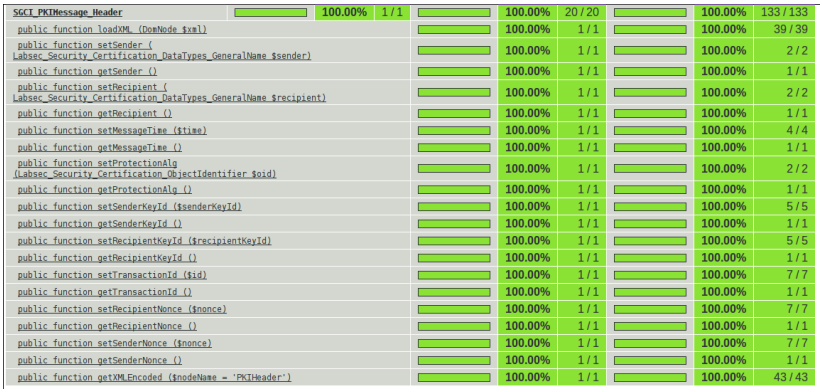
Todas as operações criptográficas realizadas nos métodos descritos acima são realizadas através da biblioteca PHP5-LibCryptoSec.

3.4 TESTES

Um dos requisitos da biblioteca é a existência de testes unitários. Cada classe implementada possui uma classe correspondente de testes. Estas são responsáveis por testar todos os métodos e todos os comportamentos que estes métodos possam apresentar, incluindo comportamentos anormais que possam resultar de parâmetros inválidos.

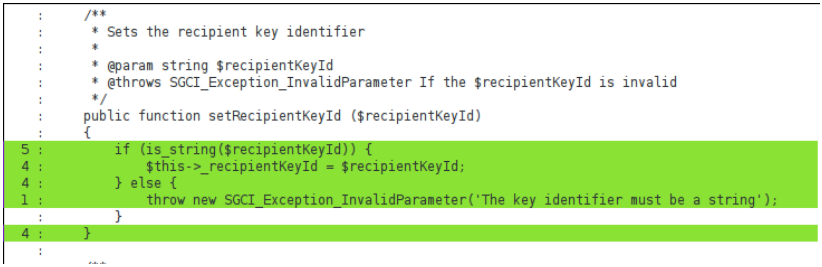
Para garantir uma boa qualidade na implementação da biblioteca, definiu-se que todo método deve ser testado em todos os cenários previsíveis. Para métodos com parâmetros, foram feitos testes passando valores válidos e um conjunto de valores inválidos. Para métodos sem parâmetros, foram feitos testes com os possíveis pré-requisitos do teste cumpridos e não cumpridos. Dessa forma garantiu-se que praticamente 100% do código da biblioteca foi executado pelos testes e que grande parte dos cenários que podem causar erros indesejáveis à aplicação foram detectados.

Para agilizar a implementação e execução dos testes a ferramenta PHPUnit foi utilizada. E para verificar a qualidade dos testes, esta ferramenta foi utilizada em conjunto com o XDebug, que gera relatórios onde é possível ver a percentagem do código testado e quantas vezes cada linha foi executada. As figuras 14 e 15 são exemplos destes relatórios.



SGCI_PKIMessage_Header	100.00%	1 / 1		100.00%	20 / 20		100.00%	133 / 133
public function loadXML (DomNode $xml)				100.00%	1 / 1		100.00%	39 / 39
public function setSender (Labsec_Security_Certification_DataTypes_GeneralName $sender)				100.00%	1 / 1		100.00%	2 / 2
public function getSender ()				100.00%	1 / 1		100.00%	1 / 1
public function setRecipient (Labsec_Security_Certification_DataTypes_GeneralName $recipient)				100.00%	1 / 1		100.00%	2 / 2
public function getRecipient ()				100.00%	1 / 1		100.00%	1 / 1
public function setMessageTime ($time)				100.00%	1 / 1		100.00%	4 / 4
public function getMessageTime ()				100.00%	1 / 1		100.00%	1 / 1
public function setProtectionAlg (Labsec_Security_Certification_ObjectIdentifier $oid)				100.00%	1 / 1		100.00%	2 / 2
public function getProtectionAlg ()				100.00%	1 / 1		100.00%	1 / 1
public function setSenderKeyId ($senderKeyId)				100.00%	1 / 1		100.00%	5 / 5
public function getSenderKeyId ()				100.00%	1 / 1		100.00%	1 / 1
public function setRecipientKeyId ($recipientKeyId)				100.00%	1 / 1		100.00%	5 / 5
public function getRecipientKeyId ()				100.00%	1 / 1		100.00%	1 / 1
public function setTransactionId ($id)				100.00%	1 / 1		100.00%	7 / 7
public function getTransactionId ()				100.00%	1 / 1		100.00%	1 / 1
public function setRecipientNonce ($nonce)				100.00%	1 / 1		100.00%	7 / 7
public function getRecipientNonce ()				100.00%	1 / 1		100.00%	1 / 1
public function setSenderNonce ($nonce)				100.00%	1 / 1		100.00%	7 / 7
public function getSenderNonce ()				100.00%	1 / 1		100.00%	1 / 1
public function getXMLEncoded ($nodeName = 'PKIHeader')				100.00%	1 / 1		100.00%	43 / 43

Figura 14: Exemplo de relatório de testes da classe PKIHeader 1



```
    :     /**
    :      * Sets the recipient key identifier
    :      *
    :      * @param string $recipientKeyId
    :      * @throws SGCI_Exception_InvalidParameter If the $recipientKeyId is invalid
    :      */
    :     public function setRecipientKeyId ($recipientKeyId)
    :     {
  5 :         if (is_string($recipientKeyId)) {
  4 :             $this->_recipientKeyId = $recipientKeyId;
  4 :         } else {
  1 :             throw new SGCI_Exception_InvalidParameter('The key identifier must be a string');
    :         }
  4 :     }
    :
```

Figura 15: Exemplo de relatório de testes da classe PKIHeader 2

3.5 DIFICULDADES DE IMPLEMENTAÇÃO

Uma das dificuldades encontradas na implementação do CMP foi encontrar uma biblioteca em PHP capaz de gerar, exportar, carregar e também validar o conteúdo do XML a partir de um XSD. Foram testadas três bibliotecas: Zend_Config_XML, SimpleXML e DomDocument.

Zend_Config_XML é uma classe do Zend Framework[1] para manipulação de arquivos XML, porém seu uso, e também sua funcionalidade, é voltada para arquivos de configuração, não satisfazendo as necessidades para esta implementação. Já as bibliotecas SimpleXML e DomDocument são padrões do php, utilizadas num propósito mais geral. A principal diferença entre elas

[1] http://framework.zend.com/manual/en/zend.config.adapters.xml.html

Características	Zend_Config_XML	SimpleXML	DomDocument
Simplicidade	Alta	Alta	Média
Suporte à validação	Não	Não	Sim
Suporte à geração	Não	Sim	Sim
Suporte à exportação	Sim	Sim	Sim
Suporte à modificação	Sim	Sim	Sim
Suporte ao carregamento	Sim	Sim	Sim

Tabela 4: Comparação das bibliotecas para manipulação de XML

é o suporte a validação dos documentos XML existente na DomDocument e ausente na primeira. A tabela 4 mostra as principais diferenças entre estas três bibliotecas.

A biblioteca que melhor atendeu as necessidades deste trabalho foi a DomDocument. O principal fator que levou à escolha desta biblioteca é o suporte à validação de arquivos XML através de um arquivo XSD, ausente nas demais.

4 INTEGRAÇÃO COM O SGCI

Na atual versão do SGCI, 1.3.7, a comunicação entre as entidades é feita apenas de forma manual. Neste modelo, o operador da AR importa a requisição de certificado, aprova a requisição e ela é disponibilizada ao operador para download. O operador então envia a requisição para a AC de forma manual, através e-mail por exemplo. Na AC, o operador deve importar a requisição recebida pela AR, aprovar ou rejeitar a requisição e enviar a resposta para a AR também de forma manual.

A partir deste trabalho, o SGCI passa a ter dois novos modelos de comunicação, ambos online. O primeiro modelo é conhecido como modelo online com AC de resposta manual, cuja única diferença do modelo offline é que o envio das mensagens entre a AR e a AC é feita de forma automática. Quando o operador da AR aprova a requisição ela é automaticamente enviada para a AC, como mostrado na figura 16.

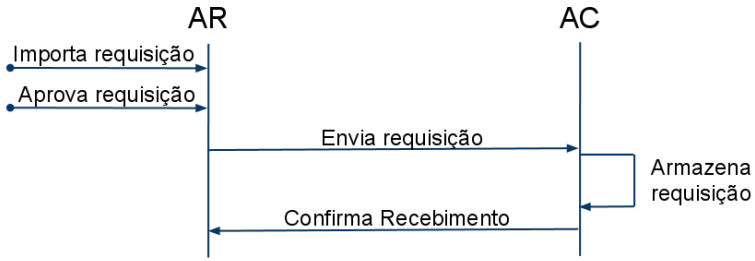


Figura 16: Modelo online com AC de resposta manual parte 1

Quando a requisição for aprovada na AC, a resposta é novamente enviada automaticamente para a AR, que pode enviar a resposta ao requisitante do certificado. Esta é a segunda parte do modelo online com AC de resposta manual, esquematizado na figura 17.

No segundo modelo, conhecido como modelo online com AC de resposta automática, a AR importa a requisição e em seguida a aprova. Ela então é enviada à AC de forma automática, assim como no modelo anterior. Quando a AC recebe a requisição, ela emite o certificado automaticamente, sem necessidade de intervenção humana, enviando a resposta para a AR. Os próximos passos ocorrem da mesma forma como no modelo anterior. A AR recebe a reposta e a encaminha para o usuário. Este modelo está esquemati-

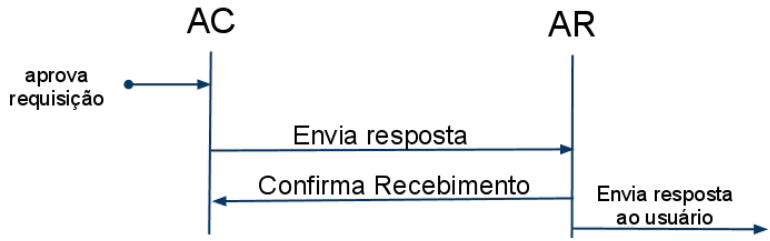


Figura 17: Modelo online com AC de resposta manual parte 2

zado na figura 18.

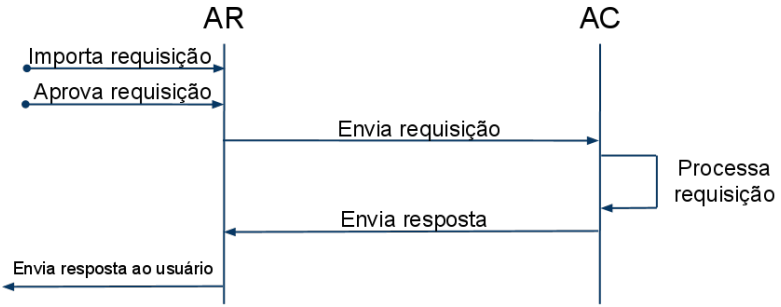


Figura 18: Modelo online com AC de resposta automática

Para adicionar o suporte ao CMP no SGCI foram necessárias algumas alterações em seu código fonte e banco de dados. As seções a seguir descrevem como foram feitas estas alterações.

4.1 ALTERAÇÕES NO BANCO DE DADOS

A primeira alteração feita no banco de dados do SGCI foi para possibilitar o armazenamento das mensagens do CMP. Para isso foi adicionada uma nova tabela, chamada *Transactions*, ilustrada na figura 19.

O significado de cada coluna desta tabela está descrito abaixo:

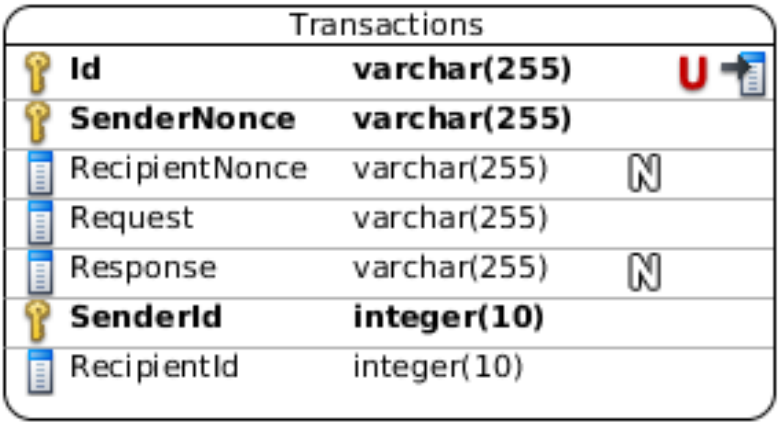


Figura 19: Diagrama de banco de dados da tabela Transactions

- Id: é o identificador da transação, refere-se ao campo *transactionId* do cabeçalho da *PKIMessage*.
- SenderNonce: é utilizado para identificar requisições duplicadas e prevenir ataques de repetição. Refere-se ao campo de mesmo nome do cabeçalho da *PKIMessage*.
- RecipientNonce: é utilizado para identificar requisições duplicadas e prevenir ataques de repetição. Refere-se ao campo de mesmo nome do cabeçalho da *PKIMessage*.
- Request: é o XML de uma *PKIMessage* referente a uma requisição do CMP. Por exemplo requisição de certificado.
- Response: é o XML de uma *PKIMessage* referente a resposta para a requisição do campo acima.
- SenderId: é o identificador da AR no banco de dados do SGCI. Esta coluna é uma chave estrangeira para a tabela *Entities*, apresentada na figura 20.
- RecipientId: é o identificador da AC no banco de dados do SGCI. Esta coluna é uma chave estrangeira para a tabela *Entities*, apresentada na figura 20.

As colunas Id, SenderNonce e SenderId representam a chave primária da tabela. Dessa forma, o SGCI permite que o mesmo identificador da transação possa ser utilizado por diferentes entidades, minimizando a possibilidade de conflitos dos identificadores. As colunas senderId e recipientId devem apontar para as entidades correspondentes, respectivamente, ao campo *sender* e *recipient* do cabeçalho da *PKIMessage*.

Com a nova tabela criada, foi necessário integrá-la ao esquema de banco de dados do SGCI. Isto foi feito colocando uma chave estrangeira nas tabelas que representam requisições de certificado e de revogação para a tabela *Transactions*, associando cada requisição com uma transação.

Por último foram feitas alterações nas tabelas que representam as ACs e ARs no banco de dados do SGCI, para possibilitar a configuração de entidades online ou offline e ACs de resposta manual ou automática. A figura 20 ilustra o diagrama de banco de dados da tabela *Entities* após as alterações

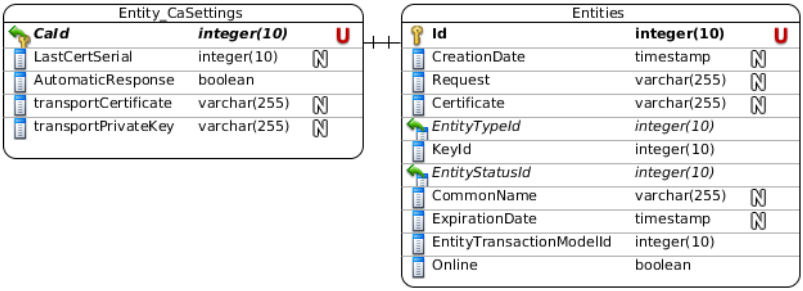


Figura 20: Diagrama de banco de dados da tabela Entities

citadas acima. As colunas *TransportCertificate* e *TransportPrivateKey* também foram adicionadas e seu uso envolve o conceito de relacionamento de confiança, que será apresentado na seção 4.2.3.

4.2 ALTERAÇÕES NO CÓDIGO FONTE

O código fonte do SGCI é divido em módulos, sendo cada módulo responsável por determinadas funcionalidades do software. Por exemplo, todas as ações possíveis por um operador de AR ficam no módulo “RaOper”. Isto facilitou ao integrar o CMP com o SGCI e também possibilitou que o SGCI não fique totalmente dependente do protocolo, sendo possível facilmente integrá-lo a outros protocolos.

Seguindo a estrutura modular do SGCI, foi adicionado um novo módulo, chamado CMP, contendo as classes responsáveis por criar as mensagens do CMP e fazer a comunicação entre diferentes SGCIs. As classes criadas, seus métodos e como é feita a comunicação entre estas classes e o restante do código do SGCI será descrito nas seções seguintes.

4.2.1 Geração das mensagens do CMP

Antes de descrever as mudanças no código do SGCI, é necessário apresentar o conceito de transação adicionado ao SGCI e quais campos das mensagens do CMP serão utilizadas pelo SGCI.

Uma transação consiste de pelo menos uma requisição e uma resposta, sendo a requisição gerada pela AR e a resposta pela AC. No SGCI, uma transação é representada pela classe *Common_Model_Transaction*, que contém os atributos referentes à tabela *Transactions*, apresentada na seção 4.1. Dentre seus atributos há a requisição e a resposta, que são representadas pela classe *SGCI_PKIMessage*.

Como descrito na seção 3.3.1, uma *PKIMessage* contém um cabeçalho, um corpo e a sua assinatura. Todas as mensagens geradas pelo SGCI possuirão os seguintes campos no cabeçalho:

- *sender*: Obtido do campo *subject* do certificado do emissor da mensagem;
- *recipient*: Obtido do campo *subject* do certificado do destinatário da mensagem;
- *senderKID*: Calculado a partir da chave pública do certificado do emissor da mensagem. Este campo é calculado de acordo com a extensão *subject key identifier* (COOPER et al., 2008);
- *recipientKID*: Calculado a partir da chave pública do certificado do destinatário da mensagem. Este campo é calculado de acordo com a extensão *subject key identifier* (COOPER et al., 2008);
- *transactionId*: Identifica uma transação. É um valor de 128 bits gerado aleatoriamente.
- *senderNonce*: Utilizado para identificar mensagens duplicadas e evitar ataques de repetição. É um valor de 128 bits gerado aleatoriamente.
- *recipientNonce*: Utilizado para identificar mensagens duplicadas e evitar ataques de repetição. É um valor de 128 bits gerado aleatoriamente.

Os campos presentes no corpo da mensagem dependem do propósito da mensagem. Para requisições de certificado, este campo é representado pela classe *SGCI_PKIMessage_Body_CertificationRequestMessage* e possuirá os seguintes campos:

- *certReqId*: É o identificador interno do SGCI para a requisição de certificado.
- *certTemplate*: Contém a chave pública e o campo *subject* da requisição de certificado.

A resposta para a requisição de certificado, apresentada acima, é representada pela classe *SGCI_PKIMessage_Body_CertificationResponseMessage* e possui os seguintes campos:

- *certReqId*: O mesmo identificador apresentado acima.
- *status*: Contém o *status* da requisição. Os possíveis *status* são: *accepted*, se o certificado foi emitido; *rejection*, se a requisição foi rejeitada; e *waiting*, se a AC recebeu a requisição mas ainda não a processou. Em caso de rejeição, ainda será informado o motivo da rejeição. Os possíveis valores deste campo serão apresentados nas próximas seções.
- *certifiedKeyPair*: Este campo só estará presente caso o *status* seja de aprovação e conterá o certificado emitido. Neste campo o CMP permite fornecer o certificado cifrado, entretanto, como o certificado é público, no SGCI, ele sempre será fornecido em claro. Ainda é possível fornecer a chave privada correspondente à chave pública presente no certificado, porém como a chave privada é gerada pelo requerente do certificado e o SGCI nunca terá a posse desta chave, o campo nunca será utilizado.

Para requisições de revogação de certificado, o corpo da *PKIMessage* é representado pela classe *SGCI_PKIMessage_Body_RevocationRequestMessage* e possuirá os seguintes campos:

- *certDetails*: Contém o número serial do certificado a ser revogado.
- *crlEntryDetails*: Contém extensões que a AR deseja que estejam presentes na LCR. Atualmente o SGCI suporte apenas a extensão *CRLBasicConstraints* (COOPER et al., 2008), que indica o motivo da revogação.

A resposta para a requisição de revogação de certificado, apresentada acima, é representada pela classe *SGCI_PKIMessage_Body_RevocationResponseMessage* e possui os seguintes campos:

- *status*: Contém o *status* da requisição. Os possíveis *status* são: *accepted*, se o certificado foi revogado; *rejection*, se a requisição foi rejeitada; e *waiting*, se a AC recebeu a requisição mas ainda não a processou. Em caso de rejeição, ainda será informado o motivo da rejeição. Os possíveis valores deste campo serão apresentados nas próximas seções.

- *revCerts*: Contém o número serial e o campo *issuer* do certificado para o qual foi solicitada a revogação. Apesar deste campo ser opcional, ele será sempre utilizado pelo SGCI para facilitar a identificar a qual certificado esta resposta é referente.

Por fim, a geração e verificação da assinatura da *PKIMessage* é feita pela classe *SGCI_PKIMessageSigner*. Os algoritmos utilizados para gerar a assinatura dependem do algoritmo da chave privada da entidade que irá gerar a assinatura e do algoritmo de hash configurado pelo usuário. Os algoritmos suportados por esta classe podem ser encontrados na seção 3.3.4.

4.2.2 Alterações para fazer a comunicação entre diferentes SGCIs

Como o SGCI é uma aplicação Web, a comunicação entre AC e AR é feita através do protocolo HTTP, seguindo as recomendações do documento *Transport Protocols for CMP* (KAUSE; PEYLO, 2011), descritas na seção 2.4.5.

A classe *CMP_ClientController* é responsável por atuar como cliente do protocolo. Ela possui o método *send*, que recebe como parâmetros uma instância de um objeto PKIMessage, descrito na seção 3.3.1, e o endereço do servidor, que é composto do IP, a porta [1] , e o diretório do servidor HTTP onde o servidor CMP se encontra. Este método cria uma nova requisição HTTP POST, seta, no cabeçalho HTTP, o *content-type* como *application/pkixcmp*, adiciona ao corpo da requisição o XML, extraído do objeto PKIMessage recebido como parâmetro, e envia a requisição ao endereço recebido como parâmetro. A resposta recebida do servidor é então retornada à função que chamou esta. Caso ocorra algum erro durante o envio da requisição, a exceção *SGCI_Exception* é lançada. Os erros que podem ocorrer é o endereço estar incorreto, o servidor estar offline ou não conseguir processar corretamente a mensagem.

A classe *CMP_ServerController* é responsável por atuar como o servidor do protocolo. Ela possui um método para receber as requisições e outro para validá-las. O primeiro método verifica as restrições definidas pelo

[1] Apesar da porta 829 estar registrada para o CMP, existem softwares, como o EJBCA, que não a utilizam, por isso há a opção de fornecer a porta.

cmpTrans (KAUSE; PEYLO, 2011) para o protocolo HTTP, isto é, se a requisição HTTP recebida é um POST, se o *content-type* está definido corretamente e se o conteúdo da mensagem é referente a alguma das mensagens do CMP suportadas pelo SGCI. Após estas validações o método verifica o tipo da mensagem e a encaminha para a classe correta. Por exemplo, para uma requisição de certificado, o método *processRequest* da classe *CMP_CertificateRequestController* é chamado. O segundo método faz as validações do cabeçalho de uma PKIMessage, listadas abaixo:

- verificar se o destinatário da requisição é uma entidade cadastrada no SGCI;
- verificar se o emissor da requisição é uma entidade cadastrada no SGCI;
- verificar se há uma relação de confiança entre o emissor e o destinatário. O conceito de relação de confiança é apresentado na seção 4.2.3;
- verificar se a mensagem já não foi importada anteriormente;
- verificar se a mensagem está assinada e se a assinatura está correta.

Caso alguma dessas validações falhe, o método retorna o código da estrutura PKIFailureInfo, definida na RFC4210 (ADAMS et al., 2005), correspondente ao erro. Caso contrário o método retorna o valor booleano *TRUE*.

4.2.3 Alterações para o estabelecimento de relação de confiança

O CMP não provê meios para a distribuição da chave pública de entidades, não sendo possível garantir que dada mensagem foi gerada por determinada entidade. Para isso foi necessário estender o CMP, adicionando quatro novas mensagens ao *PKIBody*, para possibilitar o estabelecimento de relações de confiança. Relação de confiança é um conceito utilizado pelo SGCI, para uma AC informar em quais ARs ela confia e aceita receber requisições e para uma AR informar para quais ACs ela pode enviar requisições e receber respostas.

Para as ACs também foi necessário gerar um novo par de chaves e um certificado para distribuir a chave pública deste novo par de chaves, chamado de certificado de transporte. A nova chave privada é utilizada para assinar as mensagens do CMP, devido ao fato de as chaves privadas das ACs terem seu uso muito restrito, devendo ser utilizado somente para assinar certificados e LCRs. O novo par de chaves e o certificado são utilizados exclusivamente para a geração e verificação das assinaturas das mensagens do CMP.

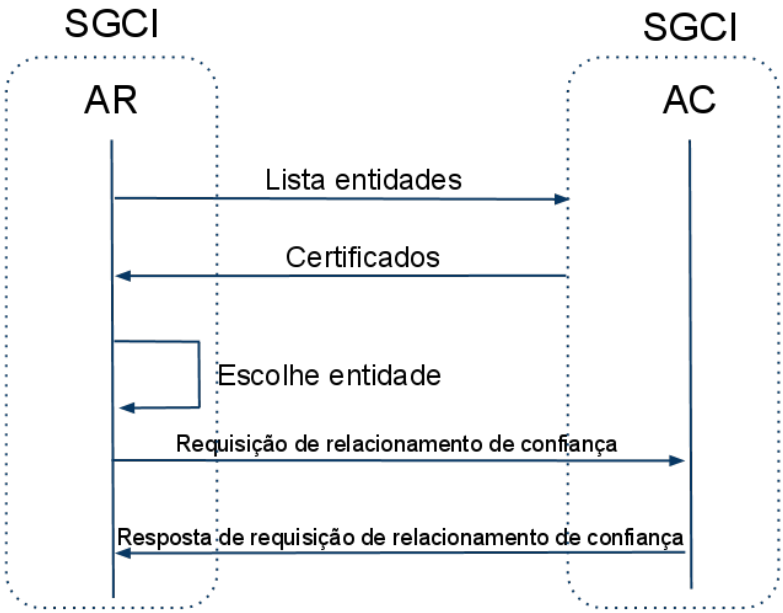


Figura 21: Estabelecimento de relacionamento de confiança no SGCI

A figura 21 mostra como funciona o estabelecimento de uma relação de confiança no SGCI. Primeiro a AR envia uma mensagem ao SGCI para listar todas as entidades cadastradas. O SGCI retorna uma lista com os certificados das entidades cadastradas e a AR escolhe com qual das entidades deseja estabelecer relacionamento de confiança. Em seguida a AR envia à entidade escolhida um pedido para o estabelecimento de relação de confiança. Neste ponto a AC precisa aprovar ou rejeitar o pedido feito pela AR e enviar uma resposta para a AR. A requisição e a resposta são assinadas com a chave privada da AR e da AC, respectivamente.

A mensagem destinada a listar os certificados das entidades é uma tag XML vazia, apresentada no trecho de código 4.1. A resposta desta mensagem é uma lista dos certificados das entidades cadastradas. Se nenhuma entidade estiver cadastrada, é retornada a tag XML vazia. O trecho de código 4.2 apresenta a estrutura ListCertsRep em XML.

```
<ListCertsReq></ListCertsReq>
```

Código Fonte 4.1: Estrutura ListCertsReq em XML

```
<ListCertsRep>
   <certificate></certificate>
</ListCertsRep>
```

Código Fonte 4.2: Estrutura ListCertsRep em XML

Para minimizar o tráfego dos certificados pela rede é possível fazer uma otimização da mensagem ListCertsReq, incluindo, por exemplo, um template de certificado, contendo informações sobre as entidades que se deseja listar. Porém esta otimização só faria sentido em um sistema com centenas ou milhares de entidades cadastradas. Como cada instância do SGCI usualmente contém apenas algumas entidades cadastradas, tal otimização traria um maior processamento para o SGCI que não resultaria em uma redução significativa nos dados trafegados pela rede. Por este motivo não foi implementado neste trabalho.

A requisição de relacionamento de confiança contém, no PKIBody, a estrutura TrustedRelReq, apresentada no trecho de código 4.3. Ela contém o endereço IP, o certificado da entidade requisitante e o certificado de transporte. O certificado de transporte só deverá estar presente se a requisição for gerada por uma AC. Neste caso este certificado será utilizado pela AR para verificar a assinatura das mensagens subsequentes entre as entidades.

```
<TrustedRelReq>
   <iPAddress></iPAddress>
   <certificate></certificate>
   <transportCertificate></transportCertificate> <!-- Opcional
       -->
</TrustedRelReq>
```

Código Fonte 4.3: Estrutura TrustedRelReq em XML

A resposta para a requisição de relacionamento de confiança contém, no PKIBody, a estrutura TrustedRelRep, apresentada no trecho de código 4.4.

Ela contém o status do pedido, descrito pela estrutura PKIStatusInfo apresentada na seção 3.2.5, o certificado da entidade e o certificado de transporte. Ambos os certificado são opcionais e só deverão estar presentes caso a relação de confiança seja aprovada. Ainda, no SGCI, o certificado de transporte só estará presente se a resposta for gerada por uma AC, pois a AR utiliza a sua própria chave privada para assinar as mensagens subsequentes que serão trocadas entre as entidades.

```
<TrustedRelRep>
   <status></status> <!-- PKIStatusInfo -->
```

```
    <certificate></certificate> <!-- Opcional -->
    <transportCertificate></transportCertificate> <!-- Opcional
        -->
</TrustedRelRep>
```

Código Fonte 4.4: Estrutura TrustedRelRep em XML

Além das estruturas em XML, foi necessário criar classes para auxiliar na geração destes XMLs pelo SGCI, semelhante ao que foi feito no capítulo 3. Foram criadas quatro classes, que implementam os métodos da classe *Body* para gerar o XML e criar um objeto a partir de um XML, além de métodos para setar e obter seus atributos, referentes aos campos das estruturas XML correspondentes a cada classe, descritas acima. O diagrama de classes da figura 22 apresenta cada uma das classes criadas, seus atributos e métodos.

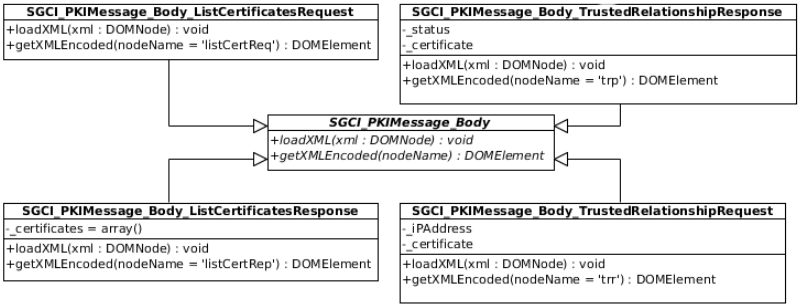


Figura 22: Diagrama de classes das classes de relacionamento de confiança

A definição ASN.1 de cada uma das estruturas criadas nesta seção, bem como a definição ASN.1 da estrutura *PKIBody* modificada, podem ser encontradas no apêndice B.

4.2.4 Alterações para requisição de certificados

Para um usuário obter um certificado através do SGCI, ele precisa primeiro encaminhar a sua requisição, no formato PKCS#10 (TURNER, 2010), à AR. O operador da AR então importa a requisição e a aprova, encaminhando a requisição para a AC responsável por emitir o certificado. O diagrama de estados representado na figura 23 mostra os diferentes estados que uma requisição pode ter no SGCI e como ocorre a transição entre elas. No momento da importação da requisição pelo operador da AR, a requisição fica num es-

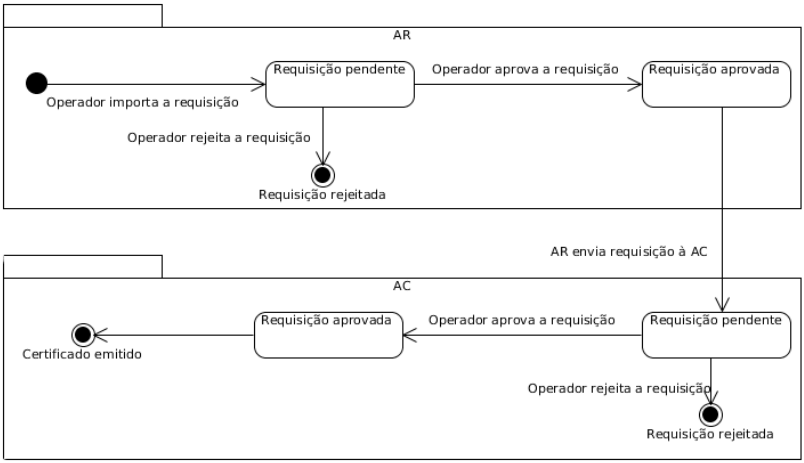


Figura 23: Diagrama de estados das requisições de certificado no SGCI

tado pendente de aprovação na AR. A partir deste estado, o operador pode rejeitar ou aprovar a requisição. Na rejeição, a requisição passa para o estado rejeitada pela AR, onde não é possível realizar mais nenhuma ação referente a requisição. Na aprovação, quando as entidades são offline, a requisição passa para o estado pendente de aprovação na AC, pois o envio da requisição para a AC não é controlado pelo SGCI. Para possibilitar a comunicação online das entidade, foi adicionado um novo estado, requisição aprovada na AR, para indicar que a requisição foi aprovada pela AR, porém ainda não foi enviada à AC. Este estado faz-se necessário caso não seja possível enviar a requisição à AC no momento da aprovação. No lado da AC, o operador pode também aprovar ou rejeitar a requisição, chegando num estado final.

Além da criação de um novo estado para as requisições de certificado, foi criada uma nova classe, chamada *CMP_CertificateRequestController*, responsável por criar a PKIMessage do CMP, encaminhar a requisição à AC, processar e validar a requisição na AC.

O diagrama de sequência representado na figura 24 mostra como é feita a troca de mensagens entre as classes já existentes do SGCI e as novas classes. O operador da AR, através da interface gráfica, envia uma mensagem ao SGCI, informando qual requisição deseja aprovar, que a encaminha à classe *RaOper_RequestController*. Esta classe faz as validações necessárias da interface, por exemplo se a requisição escolhida pelo administrador está cadastrada, e solicita à classe *CMP_CertificateRequestController* a cri-

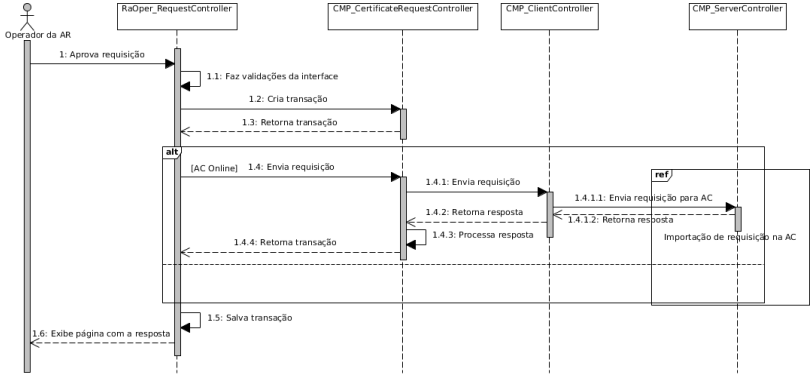


Figura 24: Diagrama de sequência da aprovação de requisição de certificado pela AR no SGCI

ação de uma nova transação. Com a transação criada, a classe *RaOper_-RequestController* verifica se a AC responsável pela emissão do certificado para a requisição aprovada é online ou offline. Se for online, a requisição é enviada à AC. A resposta da AC é processada pela classe *CMP_Certificate-RequestController*, anexada na transação e enviada de volta à classe *RaOper-_RequestController*. Se a AC for offline, nada é feito e o fluxo segue normalmente. O próximo passo é salvar a transação no banco de dados e retornar uma resposta ao operador da AR. Se a AC for online, o usuário é notificado se a requisição foi ou não enviada com sucesso, se for offline, um link para fazer o download da requisição aprovada é apresentado ao operador.

No lado da AC, a classe *CMP_ServerController* recebe, valida, verifica o tipo da requisição e a encaminha para a classe correta fazer o processamento da mesma. Neste caso é uma requisição de certificado e é enviada à classe *CMP_CertificateRequestController*, que faz as seguintes validações na requisição:

- Verifica se a chave pública do requerente do certificado está presente e se não existe um certificado emitido para esta chave.
- Verifica se o campo *subject* da requisição de certificado está presente.

Feita a validação dos dados, uma nova transação é criada na AC, correspondente à mesma transação criada anteriormente na AR, e, se a AC for de resposta automática, o certificado é emitido automaticamente. O próximo passo é a geração da resposta a ser enviada para a AR. Se a AC for de resposta

automática o certificado emitido é anexado à resposta, caso contrário a resposta enviada à AR somente indica que a AC recebeu a requisição e a mesma está aguardando aprovação do operador da AC. O diagrama de sequência representado na figura 25 mostra como é feita a troca de mensagens entre as

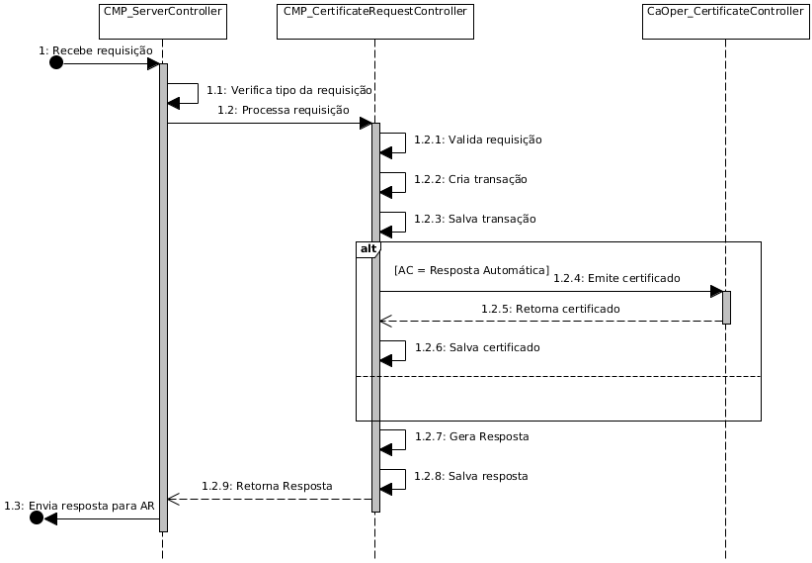


Figura 25: Diagrama de sequência da importação de requisição de certificado pela AC no SGCI

classes no lado da AC.

4.2.5 Alterações para requisição de revogação de certificados

O processo de revogação de certificado no SGCI é semelhante ao de emissão de certificado. O diagrama de estados representado na figura 26 mostra os diferentes estados que uma requisição de revogação pode ter no SGCI e como ocorre a transição entre elas. Na revogação de certificado, a requisição não possui um estado pendente na AR, pois, ao contrário da requisição de certificado, a requisição de revogação não é importada na AR. A AR solicita diretamente a revogação de um certificado e o primeiro estado da requisição é requisição aprovada na AR. Após a requisição ser enviada à AC, ela passa para o estado requisição pendente na AC. Neste momento a requi-

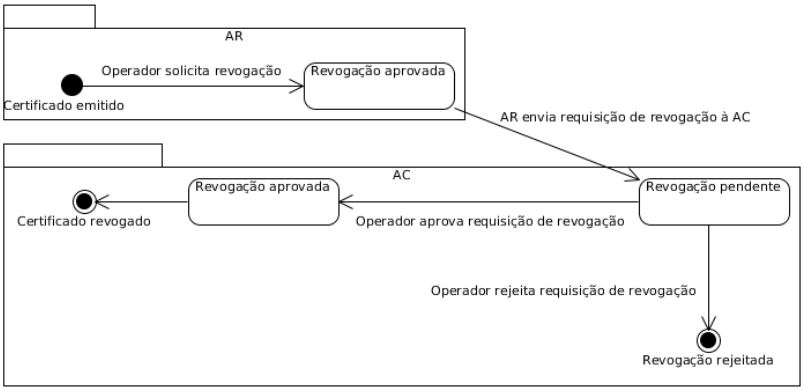


Figura 26: Diagrama de estados das requisições de revogação no SGCI

sição pode ser rejeitada ou aprovada. Se for rejeitada ela passa para o estado requisição rejeitada na AC, e se for aprovada passa para o estado requisição aprovada na AC e seu certificado passa para o estado de certificado revogado.

Para revogar um certificado, o operador da AR solicita a revogação a partir da interface gráfica. Esta solicitação é então encaminha à classe *RaOper_CertificateController* que faz as validações necessárias da interface, listadas abaixo:

- verificar se o certificado para o qual o operador está pedindo a revogação é um certificado cadastrado;
- verificar se o certificado está valido. Isto é, se não foi revogado anteriormente e se ainda não expirou;
- verificar se existe um relacionamento de confiança entre a AR e a AC que emitiu o certificado.

Após as validações, esta classe envia a mensagem para criar uma nova transação à classe *CMP_RevocationRequestController*. A transação e a PKIMessage referente ao pedido de revogação de certificado são criadas, a PKIMessage é anexada à transação e a transação é retornada. Em seguida a classe *RaOper_CertificateController* verifica se a AC é online e, em caso positivo, encaminha a transação para que a requisição de revogação seja enviada à AC, recebendo a resposta da AC. Por último a resposta é enviada ao usuário pela classe *RaOper_CertificateController*. Se a AR for online, o usuário é notificado se a requisição foi ou não enviada com sucesso, se for offline, um link

para fazer o download da requisição aprovada é apresentado ao operador. A requisição de revogação de certificado na AR está representada no diagrama se sequência da figura 27.

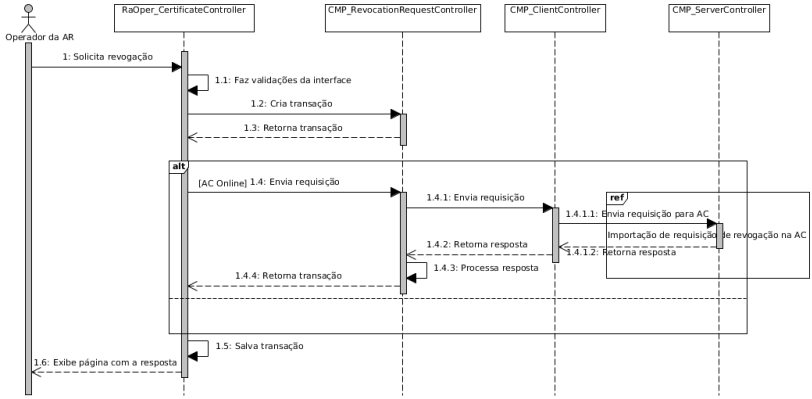


Figura 27: Diagrama de sequência da solicitação de revogação de certificado pela AR no SGCI

No lado da AC, a classe *CMP_ServerController* recebe, valida, verifica o tipo da requisição e a encaminha para a classe correta fazer o processamento da mesma. Neste caso é uma requisição de revogação de certificado e é enviada à classe *CMP_RevocationRequestController*, que faz as seguintes validações na requisição:

- verificar se o certificado para o qual foi pedido a revogação é um certificado emitido pela AC que recebeu a requisição;
- Verificar se o certificado não foi revogado anteriormente e se ainda não expirou.

Feita a validação dos dados, uma nova transação é criada na AC, correspondente à mesma transação criada anteriormente na AR, e, se a AC for de resposta automática, o certificado é revogado automaticamente. O próximo passo é a geração da resposta a ser enviada para a AR. Se a AC for de resposta automática a resposta indicará que o certificado foi revogado, caso contrário indicará que a AC recebeu a requisição e a mesma está aguardando aprovação do operador da AC. O diagrama de sequência representado na figura 28 mostra como é feita a troca de mensagens entre as classes no lado da AC.

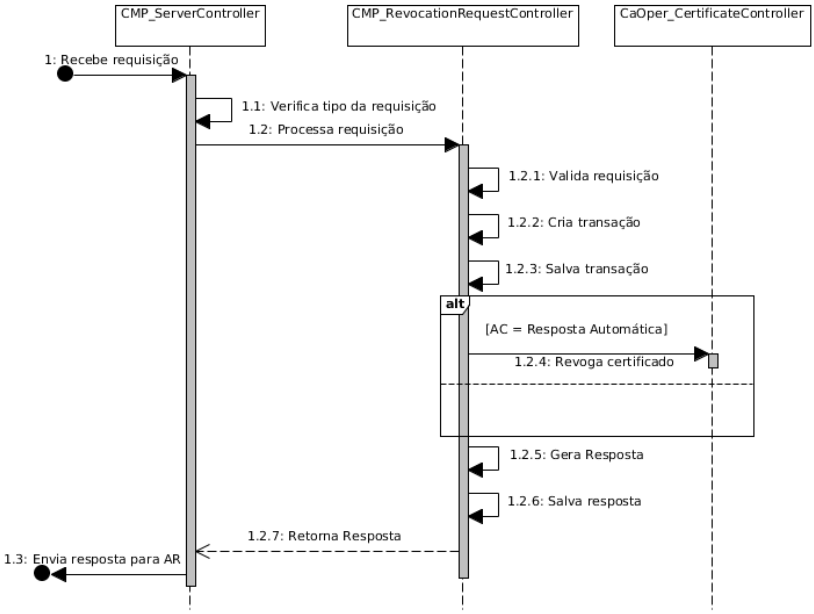


Figura 28: Diagrama de sequência da importação de requisição de certificado pela AC no SGCI

5 CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo estudar e analisar protocolos para a comunicação entre Autoridades Certificadoras e de Registro presentes na literatura, além da implementação de um protocolo para o Sistema de Gerenciamento de Certificados da Infraestrutura de Chaves Públicas para Ensino e Pesquisa (SGCI). Foram analisados os protocolos *Certificate Management over CMS* (CMC) e *Certificate Management Protocol* (CMP). O protocolo escolhido para implementar baseia-se no CMP e possui as seguintes funcionalidades: requisição para emissão e revogação de certificados; requisição para estabelecimento de relacionamento de confiança; e *pollings* para obter o estado das requisições pendentes. Ele foi escolhido, pois o CMP possui uma variedade maior de operações que podem ser realizadas, além de não depender de implementações externas.

Neste trabalho foi proposto o uso de XML para codificar as mensagens do CMP, pois o XML é uma linguagem amplamente utilizada para a troca de informações entre diferentes sistemas, além de ser simples de implementar e fornecer uma representação textual, legível para humanos. No capítulo 3 foi descrito como foi feito o mapeamento de cada estrutura ASN.1 utilizada neste trabalho para XML. Além do mapeamento das estruturas para XML, foram criadas regras, utilizando a linguagem XSD, para verificar se dado arquivo XML representa corretamente uma mensagem do CMP suportada pela implementação deste trabalho.

O capítulo 3 também descreve a implementação de uma biblioteca para facilitar a geração e manipulação das mensagens do CMP. Esta biblioteca foi implementada na linguagem PHP, orientada a objetos, seguindo as práticas de programação de Desenvolvimento Orientado a Testes (TDD, do inglês *Test Driven Development*) (BECK, 2002) e *Clean Code* (MARTIN, 2009), a fim de garantir uma alta qualidade no código desenvolvido.

Por fim, o capítulo 4 apresentou as alterações feitas no SGCI para suportar o CMP. Foram necessárias alterações no banco de dados e no código fonte. No banco de dados foi criada uma nova tabela para representar as transações do CMP, além da alteração de tabelas já existentes, adicionando novas colunas para possibilitar a configuração de entidades *online* e *offline* e de ACs de resposta automática e manual. No código fonte foi criada uma camada para fazer a interface entre o SGCI e o CMP. Ainda, foi necessário estender o CMP para suportar o conceito de relacionamento de confiança, utilizado pelo SGCI para configurar com quais ACs uma AR pode se comunicar e vice-versa. A versão do SGCI escolhida para fazer a integração é uma versão em desenvolvimento e não ainda possui implementadas todas as ope-

rações abordadas neste trabalho, como a revogação de certificados. Portanto não foi possível fazer a integração completa do CMP ao SGCI. Entretanto, não será necessário fazer novas implementações relacionadas ao protocolo CMP quando forem implementas novas operações do SGCI. Toda a estrutura necessária já foi prevista durante a elaboração deste trabalho.

5.1 TRABALHOS FUTUROS

Com as principais estruturas do CMP e a camada para o transporte implementadas, propõe-se a implementação das operações suportadas pelo CMP que não foram abordadas neste trabalho, como a substituição do par de chaves de ACs, e a integração das mesmas ao SGCI. A implementação destas operações aumentaria a flexibilidade do *software*, além de tornar as operações mais transparentes ao usuário final.

Outro trabalho futuro proposto, relacionado com as limitações deste trabalho, é o estudo de como estender o protocolo implementado para realizar a comunicação entre usuário final e AR, através da implementação de uma interface pública, chamada módulo público. A existência de um módulo público no SGCI automatizaria a comunicação entre o requerente do certificado e a AR.

Propõe-se ainda a formalização das estruturas adicionadas ao CMP e do uso de XML para a codificação das mensagens, não previsto pelo CMP. Tais formalizações são importantes para garantir que as alterações feitas não afetam o funcionamento do CMP e a interoperabilidade com outros sistemas que utilizem o CMP.

Por fim, propõe-se o estudo de como aumentar a segurança das chaves privadas das entidades *online*. Quando uma entidade é configurada para funcionar de forma *online*, sua chave privada necessita ficar disponível para o uso do SGCI. Este é um problema conhecido na literatura, porém ainda sem solução. Algumas soluções sugeridas é o uso de *tokens* criptográficos ou de um *Trusted Platform Module* (TPM).

REFERÊNCIAS BIBLIOGRÁFICAS

ADAMS, C.; FARRELL, S. *Internet X.509 Public Key Infrastructure Certificate Management Protocols*. IETF, mar. 1999. RFC 2510 (Proposed Standard). (Request for Comments, 2510). Obsoleted by RFC 4210. <http://www.ietf.org/rfc/rfc2510.txt>.

ADAMS, C. et al. *Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)*. IETF, set. 2005. RFC 4210 (Proposed Standard). (Request for Comments, 4210). <http://www.ietf.org/rfc/rfc4210.txt>.

BARKER, E.; ROGINSKY, A. *Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths*. jan 2011. <http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf>.

BECK, K. *Test Driven Development: By Example*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0321146530.

CALLAS, J. et al. *OpenPGP Message Format*. IETF, nov. 2007. RFC 4880 (Proposed Standard). (Request for Comments, 4880). Updated by RFC 5581. <http://www.ietf.org/rfc/rfc4880.txt>.

CollabNet, Inc. *Subversion*. 2011. <http://subversion.apache.org/>.

COOPER, D. et al. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. IETF, maio 2008. RFC 5280 (Proposed Standard). (Request for Comments, 5280). <http://www.ietf.org/rfc/rfc5280.txt>.

Derick Rethans. *Xdebug - Debugger and Profiler Tool for PHP*. 2011. <http://www.xdebug.org/>.

DIFFIE, W.; HELLMAN, M. E. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22, n. 6, p. 644–654, 1976.

Digital Data Security Limited. *Cryptlib Security Software*. 2011. <http://www.cryptlib.com/>.

HOUSLEY, R. *Cryptographic Message Syntax (CMS)*. IETF, set. 2009. RFC 5652 (Standard). (Request for Comments, 5652). <http://www.ietf.org/rfc/rfc5652.txt>.

HOUSLEY, R.; POLK, T. *Planning for PKI*. [S.l.]: Wiley Computer Publishing, 2001.

ITU-T. *Information technology – ASN.1 encoding rules: XML Encoding Rules (XER)*. [S.l.], dez. 2001.

ITU-T. *Abstract Syntax Notation number One*. 2011. <http://www.itu.int/ITU-T/asn1/introduction/index.htm>.

JOHNSON, D.; MENEZES, A.; VANSTONE, S. The Elliptic Curve Digital Signature Algorithm (ECDSA). *International Journal of Information Security*, v. 1, n. 1, p. 36–63, 2001. <http://dx.doi.org/10.1007/s102070100002>.

JONSSON, J.; KALISKI, B. *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1*. IETF, fev. 2003. RFC 3447 (Informational). (Request for Comments, 3447). <http://www.ietf.org/rfc/rfc3447.txt>.

KALISKI, B. *PKCS #7: Cryptographic Message Syntax Version 1.5*. IETF, mar. 1998. RFC 2315 (Informational). (Request for Comments, 2315). <http://www.ietf.org/rfc/rfc2315.txt>.

KAUSE, T.; PEYLO, M. *Internet X.509 Public Key Infrastructure – Transport Protocols for CMP*. IETF, abr. 2011. (Internet-Draft). <http://datatracker.ietf.org/doc/draft-ietf-pkix-cmp-transport-protocols/>.

KONFELDER, L. *A method for certification*. Cambridge, Massachusetts, may 1978.

LabSEC. *LibCryptoSec*. 2011. <https://projetos.labsec.ufsc.br/libcryptosec>.

LabSEC. *PHP5-LibCryptoSec*. 2011. <https://projetos.labsec.ufsc.br/php5-libcryptosec>.

Martin Peylo. *CMP for OpenSSL*. 2011. <http://sourceforge.net/apps/mediawiki/cmpforopenssl/index.php>.

MARTIN, R. C. *Clean Code: A handbook of agile software craftsmanship*. [S.l.]: Prentice Hall, 2009.

NIST. Federal Information Processing Standards Publication 46, *Data Encryption Standart (DES)*. 1977.

NIST. Technical Report, *Advanced encryption standard (AES)*. 2001.

PRITIKIN, M.; NOURSE, A.; VILHUBER, J. *Cisco Systems' Simple Certificate Enrollment Protocol*. IETF, mar. 2011. (Internet-Draft). <http://datatracker.ietf.org/doc/draft-nourse-scep/>.

SCHAAD, J. *Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF)*. IETF, set. 2005. RFC 4211 (Proposed Standard). (Request for Comments, 4211). <http://www.ietf.org/rfc/rfc4211.txt>.

SCHAAD, J.; MYERS, M. *Certificate Management over CMS (CMC)*. IETF, jun. 2008. RFC 5272 (Proposed Standard). (Request for Comments, 5272). <http://www.ietf.org/rfc/rfc5272.txt>.

SCHAAD, J.; MYERS, M. *Certificate Management over CMS (CMC): Transport Protocols*. IETF, jun. 2008. RFC 5273 (Proposed Standard). (Request for Comments, 5273). <http://www.ietf.org/rfc/rfc5273.txt>.

SCHNEIER, B. *Applied cryptography (2nd ed.): protocols, algorithms, and source code in C*. New York, NY, USA: John Wiley & Sons, Inc., 1996. ISBN 0-471-11709-9.

Sebastian Bergmann. *The PHP Unit Testing framework*. 2011. <http://www.phpunit.de/manual/current/en/index.html>.

STALLINGS, W. *Cryptography and Network Security (4th Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2005. ISBN 0131873164.

The OpenSSL Project. *OpenSSL: The Open Source toolkit for SSL/TLS*. 2011. <http://www.openssl.org/>.

The PHP Group. *PHP: Hypertext Preprocessor*. 2011. <http://www.php.net/>.

TURNER, S. *The application/pkcs10 Media Type*. IETF, ago. 2010. RFC 5967 (Informational). (Request for Comments, 5967). <http://www.ietf.org/rfc/rfc5967.txt>.

VIGIL, M. A. G. *Autoridade Certificadora de Correio Eletrônico*. [S.l.], jul. 2007. <http://projetos.inf.ufsc.br/arquivos_projetos/projeto_497/TCC-AC-Correio.pdf>.

W3C. *Extensible Markup Language (XML)*. nov 2008. <http://www.w3.org/TR/2008/REC-xml-20081126/>.

Zend Technologies. *Zend Framework*. 2011. <http://framework.zend.com/>.

ZORN, G. *PPP LCP Internationalization Configuration Option*. IETF, jan. 1999. RFC 2484 (Proposed Standard). (Request for Comments, 2484). <http://www.ietf.org/rfc/rfc2484.txt>.

APÊNDICE A – XSD para validação das estruturas das mensagens do CMP

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

        <xs:element name="PKIMessage">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="header"/>
                                <xs:element ref="body"/>
                                <xs:element name="protection"
                                    type="xs:string" minOccurs=
                                    "0"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>

        <xs:element name="header" substitutionGroup="PKIHeader"/
            >

        <xs:element name="body">
                <xs:complexType>
                        <xs:choice>
                                <xs:element ref="ir"/>
                                <xs:element ref="ip"/>
                                <xs:element ref="cr"/>
                                <xs:element ref="cp"/>
                                <xs:element ref="rr"/>
                                <xs:element ref="rp"/>
                                <xs:element ref="listCertReq"/>
                                <xs:element ref="listCertRep"/>
                                <xs:element ref="pollReq"/>
                                <xs:element ref="pollRep"/>
                                <xs:element ref="trr"/>
                                <xs:element ref="trp"/>
                        </xs:choice>
                </xs:complexType>
        </xs:element>

        <xs:element name="PKIHeader">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="pvno"/>
                                <xs:element ref="sender"/>
                                <xs:element ref="recipient"/>
                                <xs:element name="messageTime"
                                    type="xs:integer" minOccurs
                                    ="0"/>
                                <xs:element ref="protectionAlg"
                                    minOccurs="0"/>
                                <xs:element name="senderKID"
                                    type="xs:string" minOccurs=
                                    "0"/>
```

```
                        <xs:element name="recipKID" type
                            ="xs:string" minOccurs="0"/
                            >
                        <xs:element name="transactionID"
                             type="xs:string" minOccurs
                            ="0"/>
                        <xs:element name="senderNonce"
                            type="xs:string" minOccurs=
                            "0"/>
                        <xs:element name="recipNonce"
                            type="xs:string" minOccurs=
                            "0"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="protectionAlg" substitutionGroup="
    AlgorithmIdentifier"/>

<xs:element name="AlgorithmIdentifier">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="algorithm"
                            type="xs:string"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="pvno">
        <xs:simpleType>
                <xs:restriction base="xs:integer">
                        <xs:enumeration value="2"/>
            </xs:restriction>
        </xs:simpleType>
</xs:element>

<xs:element name="sender" substitutionGroup="GeneralName
    "/>
<xs:element name="recipient" substitutionGroup="
    GeneralName"/>

<xs:element name="GeneralName">
        <xs:complexType>
                <xs:choice>
                        <xs:element ref="otherName"/>
                        <xs:element name="rfc822Name"
                            type="xs:string"/>
                        <xs:element name="dNSName" type=
                            "xs:string"/>
                        <xs:element ref="directoryName"/
                            >
```

```
                        <xs:element name="
                            uniformResourceIdentifier"
                            type="xs:string"/>
                        <xs:element name="iPAddress"
                            type="xs:string"/>
                        <xs:element name="registeredID"
                            type="xs:string"/>
                </xs:choice>
        </xs:complexType>
</xs:element>

<xs:element name="otherName" substitutionGroup="
    OtherName"/>

<xs:element name="OtherName">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="type-id" type=
                            "xs:string"/>
                        <xs:element name="value" type="
                            xs:string"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="directoryName" substitutionGroup="
    RDNSequence"/>

<xs:element name="cr" substitutionGroup="CertReqMessages
    "/>
<xs:element name="ir" substitutionGroup="CertReqMessages
    "/>

<xs:element name="CertReqMessages">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="CertReqMsg"
                            maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="CertReqMsg">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="certReq"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="certReq" substitutionGroup="
    CertRequest"/>
```

```
<xs:element name="CertRequest">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="certReqId"
                            type="xs:integer"/>
                        <xs:element ref="certTemplate"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="certTemplate" substitutionGroup="
    CertTemplate"/>

<xs:element name="CertTemplate">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="version"
                            minOccurs="0"/>
                        <xs:element name="serialNumber"
                            minOccurs="0" type="
                            xs:integer"/>
                        <xs:element ref="subject"
                            minOccurs="0"/> <!--
                            Optional -->
                        <xs:element name="publicKey"
                            minOccurs="0" type="
                            xs:string"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="version" substitutionGroup="Version"/>

<xs:element name="Version">
        <xs:simpleType>
                <xs:restriction base="xs:integer">
                        <xs:enumeration value="0"/> <!--
                             v1 -->
                        <xs:enumeration value="1"/> <!--
                             v2 -->
                        <xs:enumeration value="2"/> <!--
                             v3 -->
            </xs:restriction>
        </xs:simpleType>
</xs:element>

<xs:element name="subject" substitutionGroup="
    RDNSequence"/>

<xs:element name="RDNSequence">
        <xs:complexType>
```

```
                <xs:sequence maxOccurs="unbounded">
                        <xs:choice>
                                <xs:element name="C"
                                    type="xs:string"/>
                                    <!-- Country -->
                                <xs:element name="L"
                                    type="xs:string"/>
                                    <!-- Locality -->
                                <xs:element name="ST"
                                    type="xs:string"/>
                                    <!-- State or
                                    province -->
                                <xs:element name="O"
                                    type="xs:string"/>
                                    <!-- Organization
                                    -->
                                <xs:element name="OU"
                                    type="xs:string"/>
                                    <!-- Organization
                                    Unit -->
                                <xs:element name="CN"
                                    type="xs:string"/>
                                    <!-- Common Name --
                                    >
                                <xs:element name="STREET
                                    " type="xs:string"/
                                    > <!-- Street
                                    Address -->
                                <xs:element name="E"
                                    type="xs:string"/>
                                    <!-- E-mail Address
                                     -->
                        </xs:choice>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="cp" substitutionGroup="CertRepMessage"
    />
<xs:element name="ip" substitutionGroup="CertRepMessage"
    />

<xs:element name="CertRepMessage">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="caPubs"
                            minOccurs="0"/>
                        <xs:element ref="response"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

```
<xs:element name="caPubs">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="certificate"
                            type="xs:string" maxOccurs=
                            "unbounded"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="response" substitutionGroup="
    CertResponse"/>

<xs:element name="CertResponse">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="certReqId"
                            type="xs:integer"/>
                        <xs:element ref="status"/>
                        <xs:element ref="
                            certifiedKeyPair" minOccurs
                            ="0"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="status" substitutionGroup="
    PKIStatusInfo"/>

<xs:element name="PKIStatusInfo">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="status" type="
                            xs:integer"/>
                        <xs:element name="statusString"
                            type="xs:string" minOccurs=
                            "0"/>
                        <xs:element name="failInfo" type
                            ="xs:integer" minOccurs="0"
                            />
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="certifiedKeyPair" substitutionGroup="
    CertifiedKeyPair"/>

<xs:element name="CertifiedKeyPair">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="certOrEncCert"/
                            >
```

```
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="certOrEncCert">
        <xs:complexType>
                <xs:choice>
                        <xs:element name="certificate"
                            type="xs:string"/>
                        <xs:element name="encryptedCert"
                             type="xs:string"/>
                </xs:choice>
        </xs:complexType>
</xs:element>

<xs:element name="rr" substitutionGroup="RevReqContent"/
    >

<xs:element name="RevReqContent">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="RevDetails"
                            maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="RevDetails">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="certDetails"/>
                        <xs:element ref="crlEntryDetails
                            " minOccurs="0"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="certDetails" substitutionGroup="
    CertTemplate"/>
<xs:element name="crlEntryDetails" substitutionGroup="
    Extensions"/>

<xs:element name="Extensions">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="Extension"
                            maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="Extension">
```

```
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="extnID" type="
                            xs:string"/>
                        <xs:element name="critical" type
                            ="xs:boolean"/>
                        <xs:element ref="extnValue"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="extnValue">
        <xs:complexType>
                <xs:choice>
                        <xs:element ref="reasonCode"/>
                </xs:choice>
        </xs:complexType>
</xs:element>

<xs:element name="reasonCode" substitutionGroup="
    ReasonCode"/>

<xs:element name="ReasonCode">
        <xs:simpleType>
                <xs:restriction base="xs:integer">
                        <xs:enumeration value="0"/>
                        <xs:enumeration value="1"/>
                        <xs:enumeration value="2"/>
                        <xs:enumeration value="3"/>
                        <xs:enumeration value="4"/>
                        <xs:enumeration value="5"/>
                        <xs:enumeration value="6"/>
                        <xs:enumeration value="8"/>
                        <xs:enumeration value="9"/>
                        <xs:enumeration value="10"/>
            </xs:restriction>
        </xs:simpleType>
</xs:element>

<xs:element name="rp" substitutionGroup="RevRepContent"/
    >

<xs:element name="RevRepContent">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="status"/>
                        <xs:element ref="revCerts"
                            minOccurs="0"/>
                        <xs:element ref="crls" minOccurs
                            ="0"/>
                </xs:sequence>
        </xs:complexType>
```

```
</xs:element>

<xs:element name="revCerts">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="CertId"
                            maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="crls">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="
                            CertificateList" type="
                            xs:string" maxOccurs="
                            unbounded"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="CertId">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="issuer"
                            minOccurs="0"/>
                        <xs:element name="serialNumber"
                            type="xs:integer" minOccurs
                            ="0"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="issuer" substitutionGroup="GeneralName
    "/>

<xs:element name="pollReq" substitutionGroup="
    PollReqContent"/>

<xs:element name="PollReqContent">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="certReqId"
                            maxOccurs="unbounded" type=
                            "xs:integer"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="pollRep" substitutionGroup="
    PollRepContent"/>
```

```
<xs:element name="PollRepContent">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="sequence"
                            maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="sequence">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="certReqId"
                            type="xs:integer"/>
                        <xs:element name="checkAfter"
                            type="xs:integer"/>
                        <xs:element name="reason"
                            minOccurs="0" type="
                            xs:string"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="trr" substitutionGroup="TrustedRelReq"
    />

<xs:element name="TrustedRelReq">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="iPAddress"
                            type="xs:string"/>
                        <xs:element name="certificate"
                            type="xs:string"/>
                        <xs:element name="
                            transportCertificate"
                            minOccurs="0" type="
                            xs:string"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="trp" substitutionGroup="TrustedRelRep"
    />

<xs:element name="TrustedRelRep">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="status"/>
                        <xs:element name="certificate"
                            minOccurs="0" type="
                            xs:string"/>
```

```
                                <xs:element name="
                                    transportCertificate"
                                    minOccurs="0" type="
                                    xs:string"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>

        <xs:element name="listCertReq">
                <xs:complexType>
                        <xs:sequence>
                                <!-- This structure is empty -->
                        </xs:sequence>
                </xs:complexType>
        </xs:element>

        <xs:element name="listCertRep">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="certificate"
                                    type="xs:string" minOccurs=
                                    "0" maxOccurs="unbounded"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>

</xs:schema>
```

Código Fonte A.1: Definição das estruturas do CMP em XSD

APÊNDICE B – Definição ASN.1 das estruturas criadas para estabelecimento de relacionamento de confiança no SGCI

B.1 DEFINIÇÃO ASN.1 DA ESTRUTURA LISTCERTSREQ

```
ListCertsReq ::= NULL
```

Código Fonte B.1: Estrutura ListCertsReq em ASN.1

B.2 DEFINIÇÃO ASN.1 DA ESTRUTURA LISTCERTSREP

```
ListCertsRep ::= SEQUENCE {
   certificate    Certificate }
```

Código Fonte B.2: Estrutura ListCertsRep em ASN.1

B.3 DEFINIÇÃO ASN.1 DA ESTRUTURA TRUSTEDRELREQ

```
TrustedRelReq ::= SEQUENCE {
   iPAddress                 OCTET STRING,
   certificate                Certificate,
   transportCertificate [0] Certificate OPTIONAL }
```

Código Fonte B.3: Estrutura TrustedRelReq em ASN1

B.4 DEFINIÇÃO ASN.1 DA ESTRUTURA TRUSTEDRELREP

```
TrustedRelRep ::= SEQUENCE {
   status                    PKIStatusInfo
   certificate          [0] Certificate OPTIONAL,
   transportCertificate [1] Certificate OPTIONAL }
```

Código Fonte B.4: Estrutura TrustedRelRep em ASN.1

B.5 DEFINIÇÃO ASN.1 DA ESTRUTURA PKIBODY APÓS A ADIÇÃO DAS ESTRUTURAS PARA RELACIONAMENTO DE CONFIANÇA

```
PKIBody ::= CHOICE {
```

```
ir          [0]  CertReqMessages,        --
    Initialization Req
ip          [1]  CertRepMessage,         --
    Initialization Resp
cr          [2]  CertReqMessages,        --
    Certification Req
cp          [3]  CertRepMessage,         --
    Certification Resp
p10cr       [4]  CertificationRequest,   --PKCS #10
    Cert. Req.
popdecc     [5]  POPODecKeyChallContent --pop
    Challenge
popdecr     [6]  POPODecKeyRespContent, --pop Response
kur         [7]  CertReqMessages,        --Key Update
    Request
kup         [8]  CertRepMessage,         --Key Update
    Response
krr         [9]  CertReqMessages,        --Key Recovery
     Req
krp         [10] KeyRecRepContent,       --Key Recovery
     Resp
rr          [11] RevReqContent,          --Revocation
    Request
rp          [12] RevRepContent,          --Revocation
    Response
ccr         [13] CertReqMessages,        --Cross-Cert.
     Request
ccp         [14] CertRepMessage,         --Cross-Cert.
     Resp
ckuann      [15] CAKeyUpdAnnContent,     --CA Key
    Update Ann.
cann        [16] CertAnnContent,         --Certificate
    Ann.
rann        [17] RevAnnContent,          --Revocation
    Ann.
crlann      [18] CRLAnnContent,          --CRL
    Announcement
pkiconf     [19] PKIConfirmContent,      --Confirmation
nested      [20] NestedMessageContent,   --Nested
    Message
genm        [21] GenMsgContent,          --General
    Message
genp        [22] GenRepContent,          --General
    Response
error       [23] ErrorMsgContent,        --Error
    Message
certConf    [24] CertConfirmContent,     --Certificate
    confirm
pollReq     [25] PollReqContent,         --Polling
    request
pollRep     [26] PollRepContent          --Polling
    response
```

```
    listCertReq [27] ListCertsReq          --List
        certificates request
    listCertRep [28] ListCertsRep          --List
        certificates response
    trr         29] TrustedRelReq          --Trusted
        Relationship Request
    trp         30] TrustedRelRep          --Trusted
        Relationship Response
    }
```

Código Fonte B.5: Estrutura PKIBody modificada em ASN.1

APÊNDICE C – Código fonte

```
<?php
class Cmp_Controller_CertificateRequestController
{

    /**
     * The translator.
     *
     * @var Zend_Translate
     */
    protected $_translate;

    /**
     * The logger.
     *
     * @var Common_Model_Log
     */
    protected $_log;

    /**
     * Constructor.
     *
     * @return void
     */
    public function __construct()
    {
        $this->_translate = Zend_Registry::get('Zend_Translate')
            ;
        $this->_log = Zend_Registry::get('log');
    }

    /**
     * Creates a transaction for the given request.
     *
     * This function should be called in the RA's side
     *
     * @param Common_Model_CertificateRequest $request  The
        certificate request
     * @param Common_Model_User_AuthUser      $user     The user
         who is creating the transaction
     * @param string                          $password The
        password to sign the PKIMessage
     *
     * @return Common_Model_Transaction
     */
    public function createTransaction
    (
        Common_Model_CertificateRequest $request,
        Common_Model_User_AuthUser $user,
        $password
    ) {
        $raEntityModel = $request->getRa();
```

```
$raRdnSequence = $raEntityModel->getCertificate()->
    getSubject();
$sender = new
    Labsec_Security_Certification_DataTypes_DirectoryName
    ();
$sender->addDirectoryName($raRdnSequence);

$caEntityModel = $request->getCa();
$caRdnSequence = $caEntityModel->getCertificate()->
    getSubject();
$recipient = new
    Labsec_Security_Certification_DataTypes_DirectoryName
    ();
$recipient->addDirectoryName($caRdnSequence);

$pkiMessageRequestHeader = new SGCI_PKIMessage_Header();
$pkiMessageRequestHeader->setSender($sender);
$pkiMessageRequestHeader->setRecipient($recipient);

$certificate = $user->getInternalCertificate();
$privateKey = $user->getInternalPrivateKey();

$privateKey = new LabsecCL_Security_Crypto_PrivateKeyCL(
    $privateKey, $password);

$keyPair = $raEntityModel->getKeyPair();
$raPrivateKey = $keyPair->getPrivateKey($certificate,
    $privateKey, $password);

$caCertificateBuilder = new
    LabsecCL_Security_Certification_CertificateBuilderCL
    ();

$caCertificateBuilder->setPublicKey($caEntityModel->
    getKeyPair()->getPublicKey());
$caAuthKeyIdentifier = $caCertificateBuilder->
    getPublicKeyInfo();
$pkiMessageRequestHeader->setRecipientKeyId(
    $caAuthKeyIdentifier);


$raCertificateBuilder = new
    LabsecCL_Security_Certification_CertificateBuilderCL
    ();
$raCertificateBuilder->setPublicKey($raEntityModel->
    getKeyPair()->getPublicKey());
$raAuthKeyIdentifier = $raCertificateBuilder->
    getPublicKeyInfo();
$pkiMessageRequestHeader->setSenderKeyId(
    $raAuthKeyIdentifier);
```

```
        $transactionModel = new Common_Model_Transaction();
        $transactionId = $transactionModel->generateId();
        $pkiMessageRequestHeader->setTransactionId(
            $transactionId);
        $intSenderNonce = uniqid();
        $senderNonce = sha1($intSenderNonce);
        $pkiMessageRequestHeader->setSenderNonce($senderNonce);

        $pkiMessageCertificateTemplate = new
            SGCI_PKIMessage_CertificateTemplate();
        $pkiMessageCertificateTemplate->setSubject($request->
            getRequest()->getSubject());
        $pkiMessageCertificateTemplate->setPublicKey($request->
            getRequest()->getPublicKey());
        $pkiMessageCertificationRequest = new
            SGCI_PKIMessage_CertificationRequestMessage();
        $pkiMessageCertificationRequest->setCertReqId($request->
            getId());
        $pkiMessageCertificationRequest->setCertTemplate(
            $pkiMessageCertificateTemplate);
        $pkiMessageRequestBody = new
            SGCI_PKIMessage_Body_CertificationRequestMessage();
        $pkiMessageRequestBody->addCertReqMsg(
            $pkiMessageCertificationRequest);

        $pkiMessageRequest = new SGCI_PKIMessage();
        $pkiMessageRequest->setHeader($pkiMessageRequestHeader);
        $pkiMessageRequest->setBody($pkiMessageRequestBody);

        $pkiMessageSigner = new SGCI_PKIMessageSigner();

        $pkiMessageRequest = $pkiMessageSigner->sign(
            $pkiMessageRequest,
            LabsecCL_Security_Crypto_MessageDigestCL::
                ALGORITHM_SHA256,
            $raPrivateKey
        );

        $transactionModel->setSender($raEntityModel);

        $transactionModel->setPKIMessageRequest(
            $pkiMessageRequest);
        $transactionModel->setRecipient($caEntityModel);
        $transactionModel->setSenderNonce($senderNonce);

        return $transactionModel;
    }

    /**
     * Sends the request for the given transaction.
     *
     * This function should be called in the RA's side.
```

```
     *
     * @param Common_Model_Transaction $transaction The
         transaction with the request to be send
     *
     * @return Common_Model_Transaction
     */
    public function sendRequest (Common_Model_Transaction
        $transaction)
    {
        $clientController = new Cmp_Controller_ClientController
            ();

        $ipAddress = $transaction->getRecipient()->getUri();
        $pkiMessage = $transaction->getPKIMessageRequest();
        $response = $clientController->send($pkiMessage,
            $ipAddress);

        $transaction->setPKIMessageResponse($response);

        return $transaction;
    }

    /**
     * Sends the response for the given transaction.
     *
     * This function should be called in the CA's side.
     *
     * @param Common_Model_Transaction $transaction The
         transaction with the response to be send
     *
     * @return void
     */
    public function sendResponse (Common_Model_Transaction
        $transaction)
    {
        $clientController = new Cmp_Controller_ClientController
            ();

        $ipAddress = $transaction->getRecipient()->getUri();
        $pkiMessage = $transaction->getPKIMessageResponse();
        $clientController->send($pkiMessage, $ipAddress);
    }

    /**
     * Process a CMP request.
     *
     * This function should be called in the CA's side.
     *
     * @param string                    $request A CMP request.
     * @param Common_Model_User_AuthUser $user The user
         responsible for importing the request
     *
```

```
     * @throws SGCI_PKIMessage_InvalidMessageException
     * @return Common_Model_Transaction The created transaction
         for the request
     */
    public function processRequest ($request,
        Common_Model_User_AuthUser $user)
    {
        $cmpController = new Cmp_Controller_ServerController();
        $valid = $cmpController->validatePKIMessage($request);
        if ($valid === true) {
            $pkiMessage = new SGCI_PKIMessage();
            $pkiMessage->loadXML($request);

            $ca = $user->getEntity();
            $header = $pkiMessage->getHeader();
            $dirNames = $header->getRecipient()->
                getDirectoryNames();
            $rdnSequence = $dirNames[0];
            $recipient = $rdnSequence->getEntries();

            $recipientKeyId = $header->getRecipientKeyId();
            $caCertificate = $ca->getCertificate();
            $publicKey = $caCertificate->getPublicKey();
            $publicKey = new
                LabsecCL_Security_Crypto_PublicKeyCL($publicKey
                );
            $cb = new
                LabsecCL_Security_Certification_CertificateBuilderCL
                ();
            $cb->setPublicKey($publicKey);
            $caKeyId = $cb->getPublicKeyInfo();
            $caSubject = $caCertificate->getSubject()->
                getEntries();
            if ($caSubject === $recipient && $caKeyId ===
                $recipientKeyId) {
                $transactionsMapper = new
                    Db_Model_Mappers_TransactionsMapper();
                $dirNames = $header->getSender()->
                    getDirectoryNames();
                $rdnSequence = $dirNames[0];
                $sender = $rdnSequence->getEntries();

                $entitiesMapper = new
                    Db_Model_Mappers_EntitiesMapper();
                $senderKeyId = $header->getSenderKeyId();
                $sender = $entitiesMapper->findByDnAndKeyId(
                    $sender, $senderKeyId);

                $transaction = new Common_Model_Transaction();
                $transaction->setId($pkiMessage->getHeader()->
                    getTransactionId());
                $transaction->setPKIMessageRequest($pkiMessage);
```

```
            $transaction->setRecipient($ca);
            $transaction->setSender($sender);
            $senderNonce = $header->getSenderNonce();
            $transaction->setSenderNonce($senderNonce);

            $transactionsMapper->save($transaction);

            $certificateRequests = $pkiMessage->getBody()->
                getCertReqMsgs();
            $certificateRequest = $certificateRequests[0];

            $template = $certificateRequest->getCertTemplate
                ();
            $subject = $template->getSubject()->getEntries()
                ;
            $requestModel = new
                Common_Model_CertificateRequest();
            $requestModel->setTemplate($template);
            $requestModel->setCa($ca);
            $requestModel->setCommonName($subject['CN'][0]);
            $requestModel->setImportDate(Zend_Date::now()->
                getTimestamp());
            $status =
                Db_Model_Mappers_CertificateRequestsMapper
                ::STATUS_PENDING_CA_APPROVAL;
            $requestModel->setStatus($status);

            $message = $this->_translate->_('Request
                imported') . '.'
                    . ' ' . $this->_translate->_('Request
                        Common Name') . ' = '
                    . $requestModel->getCommonName() . '.';

            $this->_log->audict($message);

            $requestsMapper = new
                Db_Model_Mappers_CertificateRequestsMapper
                ();
            $requestsMapper->save($requestModel);

            return $transaction;
        } else {
            $message = 'Your CA is not the recipient of the
                message';
            throw new
                SGCI_PKIMessage_InvalidMessageException(
                $message);
        }
    } else {
        $message = $this->_convertCodeToMessage($valid);
        throw new SGCI_PKIMessage_InvalidMessageException(
            $message, $valid);
```

```
    }
}

/**
 * Converts a PKIFailureInfo code into a human readable
     message
 *
 * @param integer $code The code of the failure
 *
 * @throws SGCI_Exception If the given code is invalid
 *
 * @return string The human readable message
 */
protected function _convertCodeToMessage ($code)
{
    switch ($code) {
        case SGCI_PKIFailureInfo::FAILURE_SIGNER_NOT_TRUSTED
            :
            $message = 'The signer of the message is not
                trusted';
        break;

        case SGCI_PKIFailureInfo::
            FAILURE_TRANSACTION_ID_IN_USE:
            $message = 'Request already imported';
        break;

        case SGCI_PKIFailureInfo::FAILURE_BAD_SENDER_NONCE:
            $message = 'Invalid Request';
        break;

        default:
            throw new SGCI_Exception('Unknown code');
        break;
    }
    return $message;
}

/**
 * Process a CMP response.
 *
 * This function should be called in the RA's side.
 *
 * @param SGCI_PKIMessage $response A CMP response.
 *
 * @return void
 */
public function processResponse (SGCI_PKIMessage $response)
{
   //TODO verify if the message was already processed
   //and if the correspond transaction exists
```

```
        $requestsMapper = new
            Db_Model_Mappers_CertificateRequestsMapper();

        $body = $response->getBody();
        $body = new
            SGCI_PKIMessage_Body_CertificationResponseMessage();

        $certReqId = $body->getCertReqId();
        $request = $requestsMapper->find($certReqId);
        if ($request !== null) {
            if ($body->getStatus()->getStatus() ===
                SGCI_PKIStatus::STATUS_REJECTION) {
                $status =
                    Db_Model_Mappers_CertificateRequestsMapper::
                    STATUS_REJECTED_BY_CA;
                $request->setStatus($status);
            } else if ($body->getStatus()->getStatus() ===
                SGCI_PKIStatus::STATUS_ACCEPTED
                   || $body->getStatus()->getStatus() ===
                       SGCI_PKIStatus::STATUS_GRANTED_WITH_MODS
            ) {
                $status =
                    Db_Model_Mappers_CertificateRequestsMapper::
                    STATUS_APPROVED_BY_CA;
                $request->setStatus($status);

                $sender = $response->getHeader()->getSender();
                $senderKeyId = $response->getHeader()->
                    getSenderKeyId();

                $entitiesMapper = new
                    Db_Model_Mappers_EntitiesMapper();
                $ca = $entitiesMapper->findByDnAndKeyId($sender,
                    $senderKeyId);

                $certificateModel = new Common_Model_Certificate
                    ();
                $certificateModel->setCa($ca);

                $certificate = $body->getCertificate();
                $subject = $certificate->getSubject();

                $certificateModel->setCertificate($certificate);
                $certificateModel->setCommonName($subject['CN'
                    ][0]);
                $certificateModel->setIssuedDate($certificate->
                    getNotBefore()->getTimestamp());
                $certificateModel->setStatus(
                    Db_Model_Mappers_CertificatesMapper::
                    STATUS_ISSUED);
```

```
            $certificatesMapper = new
                Db_Model_Mappers_CertificatesMapper();
            $certificatesMapper->save($certificateModel);
        }
        $requestsMapper->save($request);
    } else {
        //TODO what to do?
    }
}

/**
 * Creates a CMP response for the given transaction.
 *
 * This function should be called in the CA's side.
 *
 * @param Common_Model_Transaction $transaction The
     transaction in which the response will be
 *                                                 generated
 *
 * @return Common_Model_Transaction
 */
public function createResponse (Common_Model_Transaction
    $transaction)
{
    $pkiMessage = new SGCI_PKIMessage();

    $header = new SGCI_PKIMessage_Header();
    //TODO preencher header

    $body = new
        SGCI_PKIMessage_Body_CertificationResponseMessage()
        ;
    //TODO get certReqId from transaction
    $body->setCertReqId($certReqId);
    //TODO set the certificate
    $body->setCertificate($certificate);

    $status = new SGCI_PKIStatusInfo();
    //TODO where to get fail info?
    $status->setFailInfo($failInfo);
    //TODO where to get status?
    $status->setStatus($status);

    $body->setStatus($status);

    $pkiMessage->setHeader($header);
    $pkiMessage->setBody($body);

    $transaction->setPKIMessageResponse($pkiMessage);

    return $transaction;
}
```

```
}
```

Código Fonte C.1: Classe Cmp_Controller_CertificateRequestController

```
<?php
/**
 * Class that represents the client layer of the CMP
 */
class Cmp_Controller_ClientController
{
    /**
     * Action that send a CMP message to its recipient.
     *
     * @param SGCI_PKIMessage $pkiMessage The message to be send
     * @param string          $ipAddress  The IP address of the
        recipient of the message
     *
     * @throws SGCI_Exception If the message was not
        successfully sent to the server
     * @return SGCI_PKIMessage
     */
    public function send (SGCI_PKIMessage $pkiMessage,
        $ipAddress)
    {
        try{
            //IpAddress must be host+port+directory
            $client = new Zend_Http_Client($ipAddress);
            $client->setRawData($pkiMessage->getXMLEncoded(), '
                application/pkixcmp');
            $response = $client->request(Zend_Http_Client::POST)
                ;

            $status = $response->getStatus();
            if (preg_match('/^2.+/', $status) === 0) {
                throw new SGCI_Exception(
                    'Could not sucessfull send the request to
                        the server.'
                    . 'Server returned the status'
                    . ' ' . $status . '.'
                );
            }

            $response = $response->getBody();
            $pkiMessage = new SGCI_PKIMessage();
            $pkiMessage->loadXML($response);
            return $pkiMessage;
        } catch (Zend_Http_Client_Adapter_Exception $e) {
            throw new SGCI_Exception($e->getMessage());
        }
    }
}
```

Código Fonte C.2: Classe Cmp_Controller_ClientController

```
<?php
/**
 * Class to handle the CMP
 *
 */
class Cmp_IndexController extends SGCI_Controller_Action
{
    /**
     * Action that receive the CMP messages and foward them to
         the right class to process them
     *
     * @return void
     */
    public function indexAction ()
    {
        //TODO how to verify the content-type?
        if ($this->getRequest()->isPost() === true) {
            $data = file_get_contents('php://input');
            try{
                $pkiMessage = new SGCI_PKIMessage();
                $pkiMessage->loadXML($data);

                $body = $pkiMessage->getBody();
                if ($body instanceof
                    SGCI_PKIMessage_Body_CertificationRequestMessage
                    ) {
                    $response = $this->
                        _processCertificateRequest($pkiMessage)
                        ;
                } else if ($body instanceof
                    SGCI_PKIMessage_Body_CertificationResponseMessage
                    ) {
                    $this->_processCertificateResponse(
                        $pkiMessage);
                } else if ($body instanceof
                    SGCI_PKIMessage_Body_RevocationRequestMessage
                    ) {
                    $response = $this->_processRevocationRequest
                        ($pkiMessage);
                } else if ($body instanceof
                    SGCI_PKIMessage_Body_RevocationResponseMessage
                    ) {
                    $this->_processRevocationResponse(
                        $pkiMessage);
                } else if ($body instanceof
                    SGCI_PKIMessage_Body_TrustedRelationshipRequest
                    ) {
                    $response = $this->_processTrustedRelReq(
```

```
                    $pkiMessage);
                } else if ($body instanceof
                    SGCI_PKIMessage_Body_TrustedRelationshipResponse
                    ) {
                    $this->_processTrustedRelRep($pkiMessage);
                }
                $this->_response->setHttpResponseCode(200);
                $this->_response->setBody($response->
                    getXMLEncoded());
            } catch(Exception $e) {
                //TODO error processing the received message
            }

        } else {
            //TODO how to send a PKIMessage error back without
                communicating with the CA/RA?
        }
    }

    /**
     * Process a CMP CertReq message
     *
     * @param SGCI_PKIMessage $request The CMP PKIMessage
     * @return SGCI_PKIMessage
     */
    protected function _processCertificateRequest(
        SGCI_PKIMessage $request)
    {
        $recipient = $request->getHeader()->getRecipient();
        $recipientKeyId = $request->getHeader()->
            getRecipientKeyId();

        $entitiesMapper = new Db_Model_Mappers_EntitiesMapper();
        $ca = $entitiesMapper->findByDnAndKeyId($recipient,
            $recipientKeyId);
        if ($ca !== null) {
            $controller = new
                Cmp_Controller_CertificateRequestController();
            //TODO where to get user?
            $transaction = $controller->processRequest($request
                ->getXMLEncoded(), $user);

            if ($ca->isAutomaticResponse() === true) {
                //TODO issue the certificate
            }
            $transaction = $controller->createResponse(
                $transaction);
            return $transaction->getPKIMessageResponse();
        } else {
            //TODO what to return? cannot return a PKIMessage
                because the recipient is unkown
        }
```

```
}

/**
 * Process a CMP CertRep message
 *
 * @param SGCI_PKIMessage $response The CMP PKIMessage
 * @return void
 */
protected function _processCertificateResponse(
    SGCI_PKIMessage $response)
{
    $recipient = $response->getHeader()->getRecipient();
    $recipientKeyId = $response->getHeader()->
        getRecipientKeyId();

    $entitiesMapper = new Db_Model_Mappers_EntitiesMapper();
    $ra = $entitiesMapper->findByDnAndKeyId($recipient,
        $recipientKeyId);
    if ($ra !== null) {
        $controller = new
            Cmp_Controller_CertificateRequestController();
        $controller->processResponse($response);
    } else {
        //TODO what to return? cannot return a PKIMessage
            because the recipient is unkown
    }
}

/**
 * Process a CMP RevReq message
 *
 * @param SGCI_PKIMessage $request The CMP PKIMessage
 * @return SGCI_PKIMessage
 */
protected function _processRevocationRequest(SGCI_PKIMessage
     $request)
{
    $recipient = $request->getHeader()->getRecipient();
    $recipientKeyId = $request->getHeader()->
        getRecipientKeyId();

    $entitiesMapper = new Db_Model_Mappers_EntitiesMapper();
    $ca = $entitiesMapper->findByDnAndKeyId($recipient,
        $recipientKeyId);
    if ($ca !== null) {
        $controller = new
            Cmp_Controller_RevocationRequestController();
        //TODO where to get user?
        $transaction = $controller->processRequest($request
            ->getXMLEncoded(), $user);

        if ($ca->isAutomaticResponse() === true) {
```

```
                //TODO revoke the certificate
            }
            $transaction = $controller->createResponse(
                $transaction);
            return $transaction->getPKIMessageResponse();
        } else {
            //TODO what to return? cannot return a PKIMessage
                because the recipient is unkown
        }
    }

    /**
     * Process a CMP RevRep message
     *
     * @param SGCI_PKIMessage $response The CMP PKIMessage
     * @return void
     */
    protected function _processRevocationResponse(
        SGCI_PKIMessage $response)
    {
        $recipient = $response->getHeader()->getRecipient();
        $recipientKeyId = $response->getHeader()->
            getRecipientKeyId();

        $entitiesMapper = new Db_Model_Mappers_EntitiesMapper();
        $ra = $entitiesMapper->findByDnAndKeyId($recipient,
            $recipientKeyId);
        if ($ra !== null) {
            $controller = new
                Cmp_Controller_RevocationRequestController();
            $controller->processResponse($response);
        } else {
            //TODO what to return? cannot return a PKIMessage
                because the recipient is unkown
        }
    }

    /**
     * Process a SGCI TrustedRelReq message
     *
     * @param SGCI_PKIMessage $request The SGCI PKIMessage
     * @return SGCI_PKIMessage
     */
    protected function _processTrustedRelReq(SGCI_PKIMessage
        $request)
    {
        //TODO
    }

    /**
     * Process a SGCI TrustedRelRep message
     *
```

```
     * @param SGCI_PKIMessage $response The SGCI PKIMessage
     * @return void
     */
    protected function _processTrustedRelRep(SGCI_PKIMessage
        $response)
    {
        //TODO
    }
}
```

Código Fonte C.3: Classe Cmp_IndexController

```
<?php
class Cmp_Controller_RevocationRequestController
{
    /**
     * Creates a transaction for the given request
     *
     * This function should be called in the RA's side.
     *
     * @param Common_Model_?                $request  The
        revocation request
     * @param Common_Model_User_AuthUser    $user     The user
        who is creating the transaction
     * @param string                        $password The
        password to sign the PKIMessage
     *
     * @return Common_Model_Transaction
     */
    public function createTransaction
    (
        $request,
        Common_Model_User_AuthUser $user,
        $password
    ) {
        $revocationRequest = new
            SGCI_PKIMessage_Body_RevocationRequestMessage();

        $certTemplate = new SGCI_PKIMessage_CertificateTemplate
            ();

        //TODO where to get serial number?
        $certTemplate->setSerialNumber($serial);
        //TODO where to get crlEntryDetails?
        $revocationRequest->addRevocationRequest($certTemplate,
            $crlEntryDetails);

        $header = new SGCI_PKIMessage_Header();

        //TODO where to get the RA?
        $ra = $request->getRa();
```

```
        $raRdnSequence = $ra->getCertificate()->getSubject();
        $sender = new
            Labsec_Security_Certification_DataTypes_DirectoryName
            ();
        $sender->addDirectoryName($raRdnSequence);

        //TODO where to get the CA?
        $ca = $request->getCa();
        $caRdnSequence = $ca->getCertificate()->getSubject();
        $recipient = new
            Labsec_Security_Certification_DataTypes_DirectoryName
            ();
        $recipient->addDirectoryName($caRdnSequence);

        $header = new SGCI_PKIMessage_Header();
        $header->setSender($sender);
        $header->setRecipient($recipient);

        $certificate = $user->getInternalCertificate();
        $privateKey = $user->getInternalPrivateKey();

        $privateKey = new LabsecCL_Security_Crypto_PrivateKeyCL(
            $privateKey, $password);

        $keyPair = $ra->getKeyPair();
        $raPrivateKey = $keyPair->getPrivateKey($certificate,
            $privateKey, $password);

        $certificateBuilder = new
            LabsecCL_Security_Certification_CertificateBuilderCL
            ();

        $certificateBuilder->setPublicKey($ca->getKeyPair()->
            getPublicKey());
        $caAuthKeyIdentifier = $certificateBuilder->
            getPublicKeyInfo();
        $header->setRecipientKeyId($caAuthKeyIdentifier);

        $certificateBuilder->setPublicKey($ra->getKeyPair()->
            getPublicKey());
        $raAuthKeyIdentifier = $certificateBuilder->
            getPublicKeyInfo();
        $header->setSenderKeyId($raAuthKeyIdentifier);

        $transactionModel = new Common_Model_Transaction();
        $transactionId = $transactionModel->generateId();
        $header->setTransactionId($transactionId);

        $senderNonce = sha1(uniqid());
        $header->setSenderNonce($senderNonce);
    }
```

```
/**
 * Sends a request for the given transaction
 *
 * This function should be called in the RA's side.
 *
 * @param Common_Model_Transaction $transaction The
     transaction with the request to be send
 *
 * @return Common_Model_Transaction
 */
public function sendRequest (Common_Model_Transaction
    $transaction)
{
    $clientController = new Cmp_Controller_ClientController
        ();

    $ipAddress = $transaction->getRecipient()->getUri();
    $pkiMessage = $transaction->getPKIMessageRequest();
    $response = $clientController->send($pkiMessage,
        $ipAddress);

    $transaction->setPKIMessageResponse($response);

    return $transaction;
}

/**
 * Sends the response for the given transaction
 *
 * This function should be called in the CA's side.
 *
 * @param Common_Model_Transaction $transaction The
     transaction with the response to be send
 *
 * @return void
 */
public function sendResponse (Common_Model_Transaction
    $transaction)
{
    $clientController = new Cmp_Controller_ClientController
        ();

    $ipAddress = $transaction->getRecipient()->getUri();
    $pkiMessage = $transaction->getPKIMessageResponse();
    $clientController->send($pkiMessage, $ipAddress);
}

/**
 * Process a CMP request.
 *
 * @param SGCI_PKIMessage $request A CMP request.
 *
```

```
     * @return void
     */
    public function processRequest (SGCI_PKIMessage $request)
    {
        //TODO must implement Revocation models on SGCI first
    }

    /**
     * Process a CMP response.
     *
     * This function should be called in the RA's side.
     *
     * @param SGCI_PKIMessage $response A CMP response.
     *
     * @return void
     */
    public function processResponse (SGCI_PKIMessage $response)
    {
       //TODO must implement Revocation models on SGCI first
    }

    /**
     * Creates a CMP response for the given transaction.
     *
     * This function should be called in the CA's side.
     *
     * @param Common_Model_Transaction $transaction The
         transaction in which the response will be
     *                                                 generated
     *
     * @return Common_Model_Transaction
     */
    public function createResponse (Common_Model_Transaction
        $transaction)
    {
        $pkiMessage = new SGCI_PKIMessage();

        $header = new SGCI_PKIMessage_Header();
        //TODO preencher header

        $body = new
            SGCI_PKIMessage_Body_RevocationResponseMessage();

        $status = new SGCI_PKIStatusInfo();
        //TODO where to get fail info?
        $status->setFailInfo($failInfo);
        //TODO where to get status?
        $status->setStatus($status);

        //TODO where to get the certificate
        $body->addRevocationResponse($status, $certificate);
```

```
        $pkiMessage->setHeader($header);
        $pkiMessage->setBody($body);

        $transaction->setPKIMessageResponse($pkiMessage);

        return $transaction;
    }
}
```

Código Fonte C.4: Classe Cmp_Controller_RevocationRequestController

```
<?php
/**
 * Class that represents the server layer of the CMP
 */
class Cmp_Controller_ServerController
{
    /**
     * Validates if the given XML is a valid PKIMessage.
     *
     * @param string $xml A string with the XML encoded of the
         PKIMessage
     *
     * @return mixed Returns true if the message is valid or a
         PKIFailureInfo if it is not valid.
     */
    public function validatePKIMessage ($xml)
    {
        try {
            $pkiMessage = new SGCI_PKIMessage();
            $pkiMessage->loadXML($xml);

            $header = $pkiMessage->getHeader();

            $transactionId = $header->getTransactionId();
            $sender = $header->getSender();

            $transactionsMapper = new
                Db_Model_Mappers_TransactionsMapper();

            $header = $pkiMessage->getHeader();
            $dirNames = $header->getSender()->getDirectoryNames
                ();
            $rdnSequence = $dirNames[0];
            $sender = $rdnSequence->getEntries();
            $header->getSenderKeyId();

            $entitiesMapper = new
                Db_Model_Mappers_EntitiesMapper();
            $sender = $entitiesMapper->findByDnAndKeyId($sender,
                 $header->getSenderKeyId());
            if ($sender === null) {
```

```
                return SGCI_PKIFailureInfo::
                    FAILURE_SIGNER_NOT_TRUSTED;
            }

            $senderNonce = $header->getSenderNonce();
            if ($senderNonce === null) {
                return SGCI_PKIFailureInfo::
                    FAILURE_BAD_SENDER_NONCE;
            }

            $transaction = $transactionsMapper->find(
                $header->getTransactionId(),
                $sender->getId(),
                $header->getSenderNonce()
            );
            if ($transaction !== null) {
                return SGCI_PKIFailureInfo::
                    FAILURE_TRANSACTION_ID_IN_USE;
            }

            //TODO Futuramente verificar se ha relacionamento de
                confianca e retornar
            // FAILURE_SIGNER_NOT_TRUSTED se nao houver

            $pkiMessageSigner = new SGCI_PKIMessageSigner();
            $publicKey = $sender->getKeyPair()->getPublicKey();

            if ($pkiMessageSigner->verify($pkiMessage,
                $publicKey) === false) {
                return SGCI_PKIFailureInfo::
                    FAILURE_BAD_MESSAGE_CHECK;
            }

            return true;
        }catch (SGCI_Exception $e) {
            return SGCI_PKIFailureInfo::FAILURE_BAD_DATA_FORMAT;
        }
    }
}
```

Código Fonte C.5: Classe Cmp_Controller_ServerController

```
<?php
/**
 * Class that implements a PKIFailureInfo defined in RFC4210.
 *
 * PKIFailureInfo ::= BIT STRING {
 * -- since we can fail in more than one way!
 * -- More codes may be added in the future if/when required.
 * badAlg               (0), -- unrecognized or unsupported
     Algorithm Identifier
```

```
* badMessageCheck      (1), -- integrity check failed (e.g.,
    signature did not verify)
* badRequest           (2), -- transaction not permitted or
    supported
* badTime              (3),
* -- messageTime was not sufficiently close to the system time,
* -- as defined by local policy
* badCertId            (4), -- no certificate could be found
    matching the provided criteria
* badDataFormat        (5), -- the data submitted has the wrong
    format
* wrongAuthority       (6),
* -- the authority indicated in the request is different from
    the
* -- one creating the response token
* incorrectData        (7), -- the requester's data is incorrect
     (for notary services)
* missingTimeStamp     (8), -- when the timestamp is missing but
     should be there (by policy)
* badPOP               (9), -- the proof-of-possession failed
* certRevoked          (10),-- the certificate has already been
    revoked
* certConfirmed        (11),-- the certificate has already been
    confirmed
* wrongIntegrity       (12),-- invalid integrity, password based
     instead of signature or vice versa
* badRecipientNonce    (13),-- invalid recipient nonce, either
    missing or wrong value
* timeNotAvailable     (14),-- the TSA's time source is not
    available
* unacceptedPolicy     (15),-- the requested TSA policy is not
    supported by the TSA.
* unacceptedExtension (16),-- the requested extension is not
    supported by the TSA.
* addInfoNotAvailable (17),
* -- the additional information requested could not be
* -- understood or is not available
* badSenderNonce       (18),-- invalid sender nonce, either
    missing or wrong size
* badCertTemplate      (19),-- invalid cert. template or missing
     mandatory information
* signerNotTrusted     (20),-- signer of the message unknown or
    not trusted
* transactionIdInUse   (21),-- the transaction identifier is
    already in use
* unsupportedVersion   (22),-- the version of the message is not
     supported
* notAuthorized        (23),
* -- the sender was not authorized to make the preceding
* -- request or perform the preceding action
* systemUnavail        (24),-- the request cannot be handled due
     to system unavailability
```

```
 * systemFailure       (25),-- the request cannot be handled due
      to system failure
 * duplicateCertReq    (26)
 * -- certificate cannot be issued because a duplicate
     certificate already exists
 * }
 *
 */
abstract class SGCI_PKIFailureInfo
{
    //unrecognized or unsupported Algorithm Identifier
    const FAILURE_BAD_ALGOTIRHM = 0;
    //integrity check failed (e.g., signature did not verify)
    const FAILURE_BAD_MESSAGE_CHECK = 1;
    //transaction not permitted or supported
    const FAILURE_BAD_REQUEST = 2;
    //messageTime was not sufficiently close to the system time,
        as defined by local policy
    const FAILURE_BAD_TIME = 3;
    //no certificate could be found matching the provided
        criteria
    const FAILURE_BAD_CERT_ID = 4;
    //the data submitted has the wrong format
    const FAILURE_BAD_DATA_FORMAT = 5;
    //the authority indicated in the request is different from
        the one creating the response token
    const FAILURE_WRONG_AUTHORITY = 6;
    //the requester's data is incorrect (for notary services)
    const FAILURE_INCORRECT_DATA = 7;
    //when the timestamp is missing but should be there (by
        policy)
    const FAILURE_MISSING_TIMESTAMP = 8;
    //the proof-of-possession failed
    const FAILURE_BAD_POP = 9;
    //the certificate has already been revoked
    const FAILURE_CERT_REVOKED = 10;
    //the certificate has already been confirmed
    const FAILURE_CERT_CONFIRMED = 11;
    //invalid integrity, password based instead of signature or
        vice versa
    const FAILURE_WRONG_INTEGRITY = 12;
    //invalid recipient nonce, either missing or wrong value
    const FAILURE_BAD_RECIPIENT_NONCE = 13;
    //the TSA's time source is not available
    const FAILURE_TIME_NOT_AVALIABLE = 14;
    //the requested TSA policy is not supported by the TSA.
    const FAILURE_UNACCEPTED_POLICY = 15;
    //the requested extension is not supported by the TSA.
    const FAILURE_UNACCEPTED_EXTENSION = 16;
    //the additional information requested could not be
        understood or is not available
    const FAILURE_ADD_INFO_NOT_AVALIABLE = 17;
```

```
    //invalid sender nonce, either missing or wrong size
    const FAILURE_BAD_SENDER_NONCE = 18;
    //invalid cert. template or missing mandatory information
    const FAILURE_BAD_CERT_TEMPLATE = 19;
    //signer of the message unknown or not trusted
    const FAILURE_SIGNER_NOT_TRUSTED = 20;
    //the transaction identifier is already in use
    const FAILURE_TRANSACTION_ID_IN_USE = 21;
    //the version of the message is not supported
    const FAILURE_UNSUPPORTED_VERSION = 22;
    //the sender was not authorized to make the preceding
        request or perform the preceding action
    const FAILURE_NOT_AUTHORIZED = 23;
    //the request cannot be handled due to system unavailability
    const FAILURE_SYSTEM_UNAVAIL = 24;
    //the request cannot be handled due to system failure
    const FAILURE_SYSTEM_FAILURE = 25;
    //certificate cannot be issued because a duplicate
        certificate already exists
    const FAILURE_DUPLICATE_CERT_REQ = 26;

    /**
     * Verifies if a given failureInfo is a correct faiulureInfo
     *
     * @param integer $failureInfo The failureInfo to be
         verified
     *
     * @return bool true if the failureInfo is correct. false
         otherwise.
     */
    public static function verifyFailure ($failureInfo)
    {
        $class = new ReflectionClass('SGCI_PKIFailureInfo');
        $constants = $class->getConstants();

        $valid = false;
        if (array_search($failureInfo, $constants, true) !==
            false) {
            $valid = true;
        }
        return $valid;
    }
}
```

Código Fonte C.6: Classe SGCI_PKIFailureInfo

```
<?php
/**
 * Class that implements a PKIMessage defined in RFC4210.
 *
 * PKIMessage ::= SEQUENCE {
 * header           PKIHeader,
```

```
 * body              PKIBody,
 * protection    [0] PKIProtection OPTIONAL,
 * extraCerts    [1] SEQUENCE SIZE (1..MAX) OF CMPCertificate
 * OPTIONAL
 * }
 */
class SGCI_PKIMessage
{
    /**
     * The header of the message.
     *
     * @var SGCI_PKIMessage_Header
     */
    protected $_header;

    /**
     * The body os the message.
     *
     * @var SGCI_PKIMessage_Body
     */
    protected $_body;

    /**
     * The signature of the message.
     *
     * Calculated from the XML-Encoded of the header and body
         structures.
     *
     * @var string
     */
    protected $_signature;

    /**
     * Loads an XML
     *
     * @param string $xml The xml of the PKIMessage
     *
     * @throws SGCI_Exception If the XML is not valid
     * @return void
     */
    public function loadXML ($xml)
    {
        $doc = new DOMDocument();

        $doc->loadXML($xml);

        // @ used to suppress warnings may generated from
            invalid XMLs
        $valid = @$doc->schemaValidateSource($this->
            _getPKIMessageXMLSchema());
        if ($valid === false) {
            throw new SGCI_Exception('Invalid XML provided');
```

```
        }

        $headerXML = $doc->getElementsByTagName('header')->item
            (0);
        $this->_header = new SGCI_PKIMessage_Header();
        $this->_header->loadXML($headerXML);

        $bodyXML = $doc->getElementsByTagName('body')->item(0);
        $bodyXML = $bodyXML->childNodes->item(0);

        $this->_body = SGCI_PKIMessage_BodyFactory::
            getPKIBodyByXML($bodyXML);

        $protectionXML = $doc->getElementsByTagName('protection'
            )->item(0);
        if ($protectionXML !== null) {
            $protection = base64_decode($protectionXML->
                nodeValue);
            $this->_signature = $protection;
        }
    }

    /**
     * Sets the header of the message
     *
     * @param SGCI_PKIMessage_Header $header The header of the
         PKIMessage
     *
     * @return void
     */
    public function setHeader (SGCI_PKIMessage_Header $header)
    {
        $this->_header = $header;
    }

    /**
     * Gets the header of the message
     *
     * @return SGCI_PKIMessage_Header The header of the
         PKIMessage
     */
    public function getHeader ()
    {
        return $this->_header;
    }

    /**
     * Sets the body of the message
     *
     * @param SGCI_PKIMessage_Body $body The body of the
         PKIMessage
     *
```

```
     * @return void
     */
    public function setBody (SGCI_PKIMessage_Body $body)
    {
        $this->_body = $body;
    }

    /**
     * Gets the body of the message
     *
     * @return SGCI_PKIMessage_Body The body of the PKIMessage
     */
    public function getBody ()
    {
        return $this->_body;
    }

    /**
     * Sets the signature of the message
     *
     * @param string $signature The signature of the PKIMessage
     *
     * @throws SGCI_Exception_InvalidParameter if the signature
         is not a string
     * @return void
     */
    public function setSignature ($signature)
    {
        if (is_string($signature) === true) {
            $this->_signature = $signature;
        } else {
            throw new SGCI_Exception_InvalidParameter(
                'signature must be a string.' . gettype(
                    $signature) . ' ' . 'provided'
            );
        }

    }

    /**
     * Returns the signature of the message or null if the
         message is not signed
     *
     * @return null | string Returns the signature of the
         PKIMessage
     * or null if the message is not signed
     */
    public function getSignature()
    {
        return $this->_signature;
    }
```

```
/**
 * Returns the XML representation of the PKIMessage as
     string
 *
 * @throws SGCI_Exception If some of the required fields are
     empty
 * @return string
 */
public function getXMLEncoded ()
{
    if ($this->_header === null) {
        throw new SGCI_Exception('header is required and
            must be provided to generate XML');
    }

    if ($this->_body === null) {
        throw new SGCI_Exception('body is required and must
            be provided to generate XML');
    }

    $xml = new DOMDocument();

    $node = $xml->createElement('PKIMessage');

    $header = $this->_header->getXMLEncoded('header');
    $header = $xml->importNode($header, true);
    $node->appendChild($header);

    $body = $this->_body->getXMLEncoded();
    $body = $xml->importNode($body, true);
    $element = $xml->createElement('body');
    $element->appendChild($body);
    $node->appendChild($element);

    if ($this->_signature !== null) {
        $signatureBase64 = base64_encode($this->_signature);
        $element = $xml->createElement('protection',
            $signatureBase64);
        $node->appendChild($element);
    }

    $xml->appendChild($node);

    // @ used to suppress warnings may generated from
        invalid XMLs
    $validSchema = @$xml->schemaValidateSource($this->
        _getPKIMessageXMLSchema());

    if ($validSchema === false) {
        throw new SGCI_Exception('Error generating XML');
    }
```

```
        return $xml->saveXML();
    }

    /**
     * Returns the XML schema for the PKIMessage structure
     *
     * @return string
     */
    protected function _getPKIMessageXMLSchema()
    {
        return file_get_contents(__DIR__ . '/PKIMessage/data/
            pkiMessageSchema.xml');
    }
}
```

Código Fonte C.7: Classe SGCI_PKIMessage

```
<?php
/**
 * Class responsible for signing the PKIMessage structure
 *
 * @access public
 */
class SGCI_PKIMessageSigner
{
    /**
     * Signs the message
     *
     * @param SGCI_PKIMessage                    $pkiMessage
         The PKIMessage to be signed
     * @param string                             $signAlgorithm
         The hash algorithm to be used to
     *
         sign the message
     * @param Labsec_Security_Crypto_PrivateKey $privateKey
         The private key to sign the message
     * @param string                             $password
         Optional,
     *
                                                        the
         password of the private key
     *
     * @throws SGCI_Exception If the header or the body of the
         message are empty
     * @return SGCI_PKIMessage the signed PKIMessage
     */
    public function sign
    (
        SGCI_PKIMessage $pkiMessage,
        $signAlgorithm,
        Labsec_Security_Crypto_PrivateKey $privateKey,
        $password = null
```

```
) {
    if ($this->_verifyPKIMessageStructure($pkiMessage) ===
        false) {
        throw new SGCI_Exception('The header and the body
            are required and must be provided');
    }

    $oid =
        Labsec_Security_Certification_ObjectIdentifierFactory
            ::getObjectIdentifierByAlgorithm(
            $privateKey->getAlgorithm(), $signAlgorithm
        );

    $pkiMessage->getHeader()->setProtectionAlg($oid);

    $hash = $this->_getHash($pkiMessage, $signAlgorithm);

    $signer = new LabsecCL_Security_Crypto_SignerCL();
    $signature = $signer->sign($signAlgorithm, $privateKey,
        $hash, $password);
    $signature = stream_get_contents($signature);

    $pkiMessage->setSignature($signature);

    return $pkiMessage;
}

/**
 * Verifies the signature of a pkiMessage
 *
 * @param SGCI_PKIMessage                    $pkiMessage The
    signed PKIMessage
 * @param Labsec_Security_Crypto_PublicKey $publicKey  The
    public key to verify the signature
 *
 * @throws SGCI_Exception If the header or the body of the
    message are empty
 * @return boolean TRUE if the signature if valid. FALSE
    otherwise
 */
public function verify
(
    SGCI_PKIMessage $pkiMessage,
    Labsec_Security_Crypto_PublicKey $publicKey
) {
    if ($this->_verifyPKIMessageStructure($pkiMessage) ===
        false) {
        throw new SGCI_Exception('The header and the body
            are required and must be provided');
    }

    $signAlgorithm = $pkiMessage->getHeader()->
```

```
            getProtectionAlg();
        $signAlgorithm = $this->_getHashAlgorithm($publicKey->
            getAlgorithm(), $signAlgorithm);

        $hash = $this->_getHash($pkiMessage, $signAlgorithm);

        $signature = $pkiMessage->getSignature();

        $tempSignatureFile = tmpfile();
        fwrite($tempSignatureFile, $signature);

        $signer = new LabsecCL_Security_Crypto_SignerCL();
        $validSignature = $signer->verify(
            $tempSignatureFile, $signAlgorithm, $hash,
                $publicKey
        );

        return $validSignature;
    }

    /**
     * Verifies if a PKIMessage contain the header and body
     *
     * @param SGCI_PKIMessage $pkiMessage The PKIMessage to be
         verified
     *
     * @return boolean TRUE if the PKIMessage strucutre is OK.
         FALSE otherwise
     */
    protected function _verifyPKIMessageStructure (
        SGCI_PKIMessage $pkiMessage)
    {
        $valid = true;
        if ($pkiMessage->getHeader() === null) {
            $valid = false;
        } else if ($pkiMessage->getBody() === null) {
            $valid = false;
        }
        return $valid;
    }

    /**
     * Gets the hash of the PKIMessage
     *
     * @param SGCI_PKIMessage $pkiMessage     The PKIMessage to
         obtain the hash
     * @param string          $signAlgorithm The hash algorithm
     *
     * @throws SGCI_Exception If the header or the body of the
         message are empty
     * @return string
     */
```

```
protected function _getHash(SGCI_PKIMessage $pkiMessage,
    $signAlgorithm)
{
    $xml = new DOMDocument();

    $header = $pkiMessage->getHeader()->getXMLEncoded('
        header');
    $header = $xml->importNode($header, true);
    $header = $xml->saveXML($header);

    $body = $pkiMessage->getBody()->getXMLEncoded();
    $body = $xml->importNode($body, true);
    $body = $xml->saveXML($body);
    $body = '<body>' . $body . '</body>';

    $digest = new LabsecCL_Security_Crypto_MessageDigestCL()
        ;
    $digest->init($signAlgorithm);
    $digest->update($header . $body);
    $hash = $digest->doFinal();

    return $hash;
}

/**
 * Gets the HASH algorithm from a keyAlgorithm and a
    signAlgorithm
 *
 * @param string $keyAlgorithm  The key algorithm of the
    $signAlgorithm
 * @param string $signAlgorithm The algorithm of a signature
 *
 * @throws SGCI_Exception If the keyAlgorithm or the
    HashAlgorithm are not supported
 * @return string
 */
protected function _getHashAlgorithm ($keyAlgorithm,
    $signAlgorithm)
{
    $signAlgorithm = $signAlgorithm->getOID();

    $hash = LabsecCL_Security_Crypto_MessageDigestCL::
        ALGORITHM_SHA1;
    $oid =
        Labsec_Security_Certification_ObjectIdentifierFactory
            ::getObjectIdentifierByAlgorithm(
            $keyAlgorithm, $hash
        );
    if ($signAlgorithm === $oid->getOID()) {
        return $hash;
    }
    $hash = LabsecCL_Security_Crypto_MessageDigestCL::
```

```
            ALGORITHM_SHA224;
        $oid =
            Labsec_Security_Certification_ObjectIdentifierFactory
                ::getObjectIdentifierByAlgorithm(
                $keyAlgorithm, $hash
            );
        if ($signAlgorithm === $oid->getOID()) {
            return $hash;
        }
        $hash = LabsecCL_Security_Crypto_MessageDigestCL::
            ALGORITHM_SHA256;
        $oid =
            Labsec_Security_Certification_ObjectIdentifierFactory
                ::getObjectIdentifierByAlgorithm(
                $keyAlgorithm, $hash
            );
        if ($signAlgorithm === $oid->getOID()) {
            return $hash;
        }
        $hash = LabsecCL_Security_Crypto_MessageDigestCL::
            ALGORITHM_SHA384;
        $oid =
            Labsec_Security_Certification_ObjectIdentifierFactory
                ::getObjectIdentifierByAlgorithm(
                $keyAlgorithm, $hash
            );
        if ($signAlgorithm === $oid->getOID()) {
            return $hash;
        }
        $hash = LabsecCL_Security_Crypto_MessageDigestCL::
            ALGORITHM_SHA512;
        $oid =
            Labsec_Security_Certification_ObjectIdentifierFactory
                ::getObjectIdentifierByAlgorithm(
                $keyAlgorithm, $hash
            );
        if ($signAlgorithm === $oid->getOID()) {
            return $hash;
        }
    }
}
```

Código Fonte C.8: Classe SGCI_PKIMessageSigner

```
<?php
/**
 * Class that implements a PKIStatus defined in RFC4210.
 *
 * PKIStatus ::= INTEGER {
 * accepted                (0), -- you got exactly what you asked
       for
 * grantedWithMods         (1),
```

```
 * -- you got something like what you asked for; the
 * -- requester is responsible for ascertaining the differences
 * rejection              (2), -- you don't get it, more
     information elsewhere in the message
 * waiting                (3),
 * -- the request body part has not yet been processed; expect
     to
 * -- hear more later (note: proper handling of this status
 * -- response MAY use the polling req/rep PKIMessages specified
 * -- in Section 5.3.22; alternatively, polling in the
     underlying
 * -- transport layer MAY have some utility in this regard)
 * revocationWarning      (4),
 * -- this message contains a warning that a revocation is
 * -- imminent
 * revocationNotification (5),
 * -- notification that a revocation has occurred
 * keyUpdateWarning       (6)
 * -- update already done for the oldCertId specified in
 * -- CertReqMsg
 * }
 */
abstract class SGCI_PKIStatus
{
    //you got exactly what you asked for
    const STATUS_ACCEPTED = 0;
    // you got something like what you asked for;
    //the requester is responsible for ascertaining the
        differences
    const STATUS_GRANTED_WITH_MODS = 1;
    //you don't get it, more information elsewhere in the
        message
    const STATUS_REJECTION = 2;
    //the request body part has not yet been processed; expect
        to hear more later
    const STATUS_WAITING = 3;
    //this message contains a warning that a revocation is
        imminent
    const STATUS_REVOCATION_WARNING = 4;
    //notification that a revocation has occurred
    const STATUS_REVOCATION_NOTIFICATION = 5;
    //update already done for the oldCertId specified in
        CertReqMsg
    const STATUS_KEY_UPDATE_WARNING = 6;

    /**
     * Verifies if a given status is a correct status
     *
     * @param integer $status The status to be verified
     *
     * @return bool true if the status is correct. false
         otherwise.
```

```
     */
    public static function verifyStatus ($status)
    {
        $class = new ReflectionClass('SGCI_PKIStatus');
        $constants = $class->getConstants();

        $valid = false;
        if (array_search($status, $constants, true) !== false) {
            $valid = true;
        }
        return $valid;
    }
}
```

Código Fonte C.9: Classe SGCI_PKIStatus

```
<?php
/**
 * Class that implements a PKIStatusInfo defined in RFC4210.
 *
 * PKIStatusInfo ::= SEQUENCE {
 * status        PKIStatus,
 * statusString  PKIFreeText     OPTIONAL,
 * failInfo      PKIFailureInfo  OPTIONAL
 * }
 *
 */
class SGCI_PKIStatusInfo
{
    /**
     * The status.
     *
     * @var integer
     */
    protected $_status;

    /**
     * Human readble string about the status.
     *
     * @var string
     */
    protected $_statusString;

    /**
     * Info about the failure.
     *
     * Present only if status is rejection.
     *
     * @var integer
     */
    protected $_failInfo;
```

```
/**
 * Loads an XML
 *
 * @param DOMNode $xml The XML to import
 *
 * @return void
 */
public function loadXML (DomNode $xml)
{
    $status = $xml->getElementsByTagName('status')->item(0);
    $this->setStatus((int) $status->nodeValue);

    $statusString = $xml->getElementsByTagName('statusString
        ')->item(0);
    if ($statusString !== null) {
        $this->setStatusString($statusString->nodeValue);
    }

    $failInfo = $xml->getElementsByTagName('failInfo')->item
        (0);
    if ($failInfo !== null) {
        $this->setFailInfo((int) $failInfo->nodeValue);
    }
}

/**
 * Sets the status
 *
 * @param integer $status The status
 *
 * @throws SGCI_Exception_InvalidParameter If the $status is
     invalid
 * @return void
 */
public function setStatus ($status)
{
    if (SGCI_PKIStatus::verifyStatus($status) === true) {
        $this->_status = $status;
    } else {
        throw new SGCI_Exception_InvalidParameter('Invalid
            status provided');
    }
}

/**
 * Gets the status
 *
 * @return integer
 */
public function getStatus ()
{
    return $this->_status;
```

```
    }

    /**
     * Sets a human readable message
     *
     * @param string $status A human readable message
     *
     * @throws SGCI_Exception_InvalidParameter If the $status is
         invalid
     * @return void
     */
    public function setStatusString ($status)
    {
        if (is_string($status) === true) {
            $this->_statusString = $status;
        } else {
            throw new SGCI_Exception_InvalidParameter(
                'status must be a string.' . ' ' . gettype(
                    $status) . ' ' . 'provided'
            );
        }
    }

    /**
     * Gets the human readable message
     *
     * @return string
     */
    public function getStatusString ()
    {
        return $this->_statusString;
    }

    /**
     * Sets a failure information about the status
     *
     * @param integer $failInfo The failure information
     *
     * @see SGCI_PKIFailureInfo
     * @throws SGCI_Exception_InvalidParameter If the failInfo
         is invalid
     * @throws SGCI_Exception If the status was not yet set or
         is not a rejection status
     * @return void
     */
    public function setFailInfo ($failInfo)
    {
        if ($this->_status === SGCI_PKIStatus::STATUS_REJECTION)
            {
            if (SGCI_PKIFailureInfo::verifyFailure($failInfo)
                === true) {
                $this->_failInfo = $failInfo;
```

```
        } else {
            throw new SGCI_Exception_InvalidParameter('
                Invalid failure info provided');
        }
    } else {
        throw new SGCI_Exception('Status is not yet set or
            is not a rejection status');
    }
}

/**
 * Returns the failInfo
 *
 * @return integer
 */
public function getFailInfo ()
{
    return $this->_failInfo;
}

/**
 * Returns the XML representation of the CertReqMessages
 *
 * @param string $nodeName The name of the root node of the
     XML
 *
 * @throws SGCI_Exception if the status is empty
 * @return DOMElement
 */
public function getXMLEncoded ($nodeName = 'PKIStatusInfo')
{
    if ($this->_status === null) {
        throw new SGCI_Exception('status is required and
            must be providaded to generate XML');
    }
    $xml = new DomDocument('1.0');
    $pkiStatusInfo = $xml->createElement($nodeName);
    $element = $xml->createElement('status', $this->_status)
        ;
    $pkiStatusInfo->appendChild($element);
    if ($this->_statusString !== null) {
        $element = $xml->createElement('statusString', $this
            ->_statusString);
        $pkiStatusInfo->appendChild($element);
    }
    if ($this->_failInfo !== null) {
        $element = $xml->createElement('failInfo', $this->
            _failInfo);
        $pkiStatusInfo->appendChild($element);
    }
    return $pkiStatusInfo;
}
```

```
}
```

Código Fonte C.10: Classe SGCI_PKIStatusInfo

```
<?php
/**
 * Class that implements a PKIBody defined in RFC4210.
 *
 * PKIBody ::= CHOICE {
 * ir       [0]  CertReqMessages,        --Initialization Req
 * ip       [1]  CertRepMessage,         --Initialization Resp
 * cr       [2]  CertReqMessages,        --Certification Req
 * cp       [3]  CertRepMessage,         --Certification Resp
 * p10cr    [4]  CertificationRequest,   --PKCS #10 Cert. Req.
 * popdecc  [5]  POPODecKeyChallContent --pop Challenge
 * popdecr  [6]  POPODecKeyRespContent, --pop Response
 * kur      [7]  CertReqMessages,        --Key Update Request
 * kup      [8]  CertRepMessage,         --Key Update Response
 * krr      [9]  CertReqMessages,        --Key Recovery Req
 * krp      [10] KeyRecRepContent,       --Key Recovery Resp
 * rr       [11] RevReqContent,          --Revocation Request
 * rp       [12] RevRepContent,          --Revocation Response
 * ccr      [13] CertReqMessages,        --Cross-Cert. Request
 * ccp      [14] CertRepMessage,         --Cross-Cert. Resp
 * ckuann   [15] CAKeyUpdAnnContent,     --CA Key Update Ann.
 * cann     [16] CertAnnContent,         --Certificate Ann.
 * rann     [17] RevAnnContent,          --Revocation Ann.
 * crlann   [18] CRLAnnContent,          --CRL Announcement
 * pkiconf  [19] PKIConfirmContent,      --Confirmation
 * nested   [20] NestedMessageContent,   --Nested Message
 * genm     [21] GenMsgContent,          --General Message
 * genp     [22] GenRepContent,          --General Response
 * error    [23] ErrorMsgContent,        --Error Message
 * certConf [24] CertConfirmContent,     --Certificate confirm
 * pollReq  [25] PollReqContent,         --Polling request
 * pollRep  [26] PollRepContent          --Polling response
 * }
 *
 */
abstract class SGCI_PKIMessage_Body
{

    /**
     * Loads an XML
     *
     * @param DOMNode $xml The XML to be loaded
     *
     * @return void
     */
     public abstract function loadXML (DOMNode $xml);

     /**
```

```
     * Returns the XML representation of the PKIBody
     *
     * @param string $nodeName The name of the root node of the
         XML
     *
     * @return DOMElement
     */
    public abstract function getXMLEncoded ($nodeName);
}
```

Código Fonte C.11: Classe SGCI_PKIMessage_Body

```
<?php
/**
 * Class that uses the Factory design pattern to create PKIBody
     structures
 *
 * @access public
 */
class SGCI_PKIMessage_BodyFactory
{
    /**
     * All types defined in the PKIBody choice
     */
    const TYPE_INITIALIZATION_REQUEST = 'ir';
    const TYPE_INITIALIZATION_RESPONSE = 'ip';
    const TYPE_CERTIFICATION_REQUEST = 'cr';
    const TYPE_CERTIFICATION_RESPONSE = 'cp';
    const TYPE_REVOCATION_REQUEST = 'rr';
    const TYPE_REVOCATION_RESPONSE = 'rp';
    const TYPE_LIST_CERTIFICATES_REQUEST = 'listCertReq';
    const TYPE_LIST_CERTIFICATES_RESPONSE = 'listCertRep';
    const TYPE_TRUSTED_RELATIONSHIP_REQUEST = 'trr';
    const TYPE_TRUSTED_RELATIONSHIP_RESPONSE = 'trp';

    /**
     * Loads an XML and returns the corresponding PKIBody
     *
     * @param DomNode $xml The XML representation of the PKIBody
     *
     * @throws SGCI_Exception If the XML does not correspond to
         a known PKIBody
     * @return SGCI_PKIMessage_Body
     */
    public static function getPKIBodyByXML (DomNode $xml)
    {
        $bodyType = $xml->nodeName;
        $body = null;

        switch ($bodyType) {
            case self::TYPE_INITIALIZATION_REQUEST:
```

```
            $body = new
                SGCI_PKIMessage_Body_InitializationRequestMessage
                ();
        break;

        case self::TYPE_INITIALIZATION_RESPONSE:
            $body = new
                SGCI_PKIMessage_Body_InitializationResponseMessage
                ();
        break;

        case self::TYPE_CERTIFICATION_REQUEST:
            $body = new
                SGCI_PKIMessage_Body_CertificationRequestMessage
                ();
        break;

        case self::TYPE_CERTIFICATION_RESPONSE:
            $body = new
                SGCI_PKIMessage_Body_CertificationResponseMessage
                ();
        break;

        case self::TYPE_REVOCATION_REQUEST:
            $body = new
                SGCI_PKIMessage_Body_RevocationRequestMessage
                ();
        break;

        case self::TYPE_REVOCATION_RESPONSE:
            $body = new
                SGCI_PKIMessage_Body_RevocationResponseMessage
                ();
        break;

        case self::TYPE_LIST_CERTIFICATES_REQUEST:
            $body = new
                SGCI_PKIMessage_Body_ListCertificatesRequest
                ();
        break;

        case self::TYPE_LIST_CERTIFICATES_RESPONSE:
            $body = new
                SGCI_PKIMessage_Body_ListCertificatesResponse
                ();
        break;

        case self::TYPE_TRUSTED_RELATIONSHIP_REQUEST:
             $body = new
                 SGCI_PKIMessage_Body_TrustedRelationshipRequest
                 ();
        break;
```

```
            case self::TYPE_TRUSTED_RELATIONSHIP_RESPONSE:
                $body = new
                    SGCI_PKIMessage_Body_TrustedRelationshipResponse
                    ();
                break;

            default:
                throw new SGCI_Exception('Body type:' . ' ' .
                    $bodyType . ' ' . 'not supported');
                break;
        }
        $body->loadXML($xml);
        return $body;
    }
}
```

Código Fonte C.12: Classe SGCI_PKIMessage_BodyFactory

```
<?php
/**
 * Class that implements a CertTemplate defined in RFC2511
 *
 * CertTemplate ::= SEQUENCE {
 * version       [0] Version                OPTIONAL,
 * serialNumber  [1] INTEGER                OPTIONAL,
 * signingAlg    [2] AlgorithmIdentifier    OPTIONAL,
 * issuer        [3] Name                   OPTIONAL,
 * validity      [4] OptionalValidity       OPTIONAL,
 * subject       [5] Name                   OPTIONAL,
 * publicKey     [6] SubjectPublicKeyInfo   OPTIONAL,
 * issuerUID     [7] UniqueIdentifier       OPTIONAL,
 * subjectUID    [8] UniqueIdentifier       OPTIONAL,
 * extensions    [9] Extensions             OPTIONAL
 * }
 *
 * OptionalValidity ::= SEQUENCE {
 * notBefore  [0] Time OPTIONAL,
 * notAfter   [1] Time OPTIONAL
 * } --at least one must be present
 *
 * Time ::= CHOICE {
 * utcTime        UTCTime,
 * generalTime    GeneralizedTime
 * }
 *
 */
class SGCI_PKIMessage_CertificateTemplate
{
    /**
     * Certificate version.
     *
```

```
     * Possible values for the version are:
     * 0 for v1
     * 1 for v2
     * 2 for v3
     *
     * @var integer
     */
    protected $_version;

    /**
     * Serial number of the certificate.
     *
     * @var integer
     */
    protected $_serialNumber;

    /**
     * Subject field of the certificate request.
     *
     * @var Labsec_Security_Certification_DataTypes_RDNSequence
     */
    protected $_subject;

    /**
     * The public key of the certificate request.
     *
     * @var Labsec_Security_Crypto_PublicKey
     */
    protected $_publicKey;

    /**
     * Loads an XML
     *
     * @param DOMNode $xml The XML to be loaded
     *
     * @return void
     */
    public function loadXML (DomNode $xml)
    {
        $version = $xml->getElementsByTagName('version')->item
            (0);
        if ($version !== null) {
            $this->setVersion((int) $version->nodeValue);
        }
        $serialNumber = $xml->getElementsByTagName('serialNumber
            ')->item(0);
        if ($serialNumber !== null) {
            $this->setSerialNumber((int) $serialNumber->
                nodeValue);
        }
        $subject = $xml->getElementsByTagName('subject')->item
            (0);
```

```
        if ($subject !== null) {
            $subjectObject = new
                Labsec_Security_Certification_DataTypes_RDNSequence
                ();
            $subjectObject->loadXML($subject);
            $this->setSubject($subjectObject);
        }
        $pubKey = $xml->getElementsByTagName('publicKey')->item
            (0);
        if ($pubKey !== null) {
            $pubKeyObject = new
                LabsecCL_Security_Crypto_PublicKeyCL($pubKey->
                nodeValue);
            $this->setPublicKey($pubKeyObject);
        }
    }

    /**
     * Sets the version of the certificate
     *
     * @param numeric $version The version of the certificate
     *
     * @throws SGCI_Exception_InvalidParameter If the parameter
         providaded is invalid
     * @return void
     */
    public function setVersion ($version)
    {
        if ((is_numeric($version) === true)
            && ($this->_isValidVersion($version) === true)
        ) {
            $this->_version = $version;
        } else {
            throw new SGCI_Exception_InvalidParameter('Invalid
                version providaded');
        }
    }

    /**
     * Verifies if the given version is a valid certificate
         version
     *
     * @param integer $version The version of the certificate
     *
     * @return boolean TRUE if the version if valid. FALSE
         otherwise.
     */
    protected function _isValidVersion ($version)
    {
        if ($version === 0 || $version === 1 || $version === 2)
            {
            return true;
```

```
        }
        return false;
    }

    /**
     * Gets the version of the certificate
     *
     * @return numeric
     */
    public function getVersion ()
    {
        return $this->_version;
    }

    /**
     * Sets the serial number of the certificate
     *
     * @param integer $serial The serial of the certificate
     *
     * @throws SGCI_Exception_InvalidParameter If the parameter
         providaded is invalid
     * @return void
     */
    public function setSerialNumber ($serial)
    {
        if (is_integer($serial) === true) {
            $this->_serialNumber = $serial;
        } else {
            throw new SGCI_Exception_InvalidParameter(
                'Parameter must be an integer.' . ' ' . gettype(
                    $serial) . ' ' .'given'
            );
        }
    }

    /**
     * Gets the serial number of the certificate
     *
     * @return integer
     */
    public function getSerialNumber ()
    {
        return $this->_serialNumber;
    }

    /**
     * Sets the subject of the certificate request
     *
     * @param
         Labsec_Security_Certification_DataTypes_RDNSequence
         $subject The subject of the
     *
```

```
     certificate
 *
 * @throws SGCI_Exception_InvalidParameter If the
     RDNSequence provided is empty
 * @return void
 */
public function setSubject (
    Labsec_Security_Certification_DataTypes_RDNSequence
    $subject)
{
    try {
        $subject->getEntries();
        $this->_subject = $subject;
    } catch (Exception $e) {
        throw new SGCI_Exception_InvalidParameter('Subject
            must have at least one entry');
    }
}

/**
 * Gets the subject of the certificate request
 *
 * @return
     Labsec_Security_Certification_DataTypes_RDNSequence
 */
public function getSubject ()
{
    return $this->_subject;
}

/**
 * Sets the subject's public key of the certificate request
 *
 * @param Labsec_Security_Crypto_PublicKey $publicKey The
     public key fo the certificate
 *
 * @return void
 */
public function setPublicKey (
    Labsec_Security_Crypto_PublicKey $publicKey)
{
    $this->_publicKey = $publicKey;
}

/**
 * Gets the subject's public key of the certificate request
 *
 * @return Labsec_Security_Crypto_PublicKey
 */
public function getPublicKey ()
{
```

```
        return $this->_publicKey;
    }

    /**
     * Verifies if the required fields of the certTemplate are
         empty
     *
     * @return boolean TRUE if all required fields are empty,
         FALSE otherwise
     */
    public function isEmpty ()
    {
        if ($this->_version === null && $this->_serialNumber ===
             null
            && $this->_subject === null && $this->_publicKey ===
                 null
        ) {
            return true;
        }
        return false;
    }

    /**
     * Returns the XML representation of the
         CertificationRequestMessage
     *
     * @param string $nodeName The name of the root node of the
         XML
     *
     * @throws SGCI_Exception if the certTemplate is empty
     * @return DOMElement
     */
    public function getXMLEncoded ($nodeName = 'CertTemplate')
    {
        if ($this->isEmpty() === true) {
            throw new SGCI_Exception(
                'At least one field must be present to generate
                    the XML'
            );
        }
        $xml = new DomDocument('1.0');
        $node = $xml->createElement($nodeName);
        if ($this->_version !== null) {
            $element = $xml->createElement('version', $this->
                _version);
            $node->appendChild($element);
        }
        if ($this->_serialNumber !== null) {
            $element = $xml->createElement('serialNumber', $this
                ->_serialNumber);
            $node->appendChild($element);
        }
```

```
        if ($this->_subject !== null) {
            $subjectXML = $this->_subject->getEntriesAsDomNode('
                subject');
            $subjectXML = $xml->importNode($subjectXML, true);
            $node->appendChild($subjectXML);
        }
        if ($this->_publicKey !== null) {
            $element = $xml->createElement('publicKey', $this->
                _publicKey->getPemEncoded());
            $node->appendChild($element);
        }
        return $node;
    }
}
```

Código Fonte C.13: Classe SGCI_PKIMessage_CertificateTemplate

```
<?php
/**
 * Class that implements a CertReqMessage defined in RFC2511.
 *
 * CertReqMsg ::= SEQUENCE {
 * certReq   CertRequest,
 * pop       ProofOfPossession  OPTIONAL,
 * -- content depends upon key type
 * regInfo   SEQUENCE SIZE(1..MAX) of AttributeTypeAndValue
     OPTIONAL }
 *
 * CertRequest ::= SEQUENCE {
 * certReqId     INTEGER,          -- ID for matching request
     and reply
 * certTemplate  CertTemplate, -- Selected fields of cert to be
      issued
 * controls      Controls OPTIONAL }  -- Attributes affecting
     issuance
 * }
 *
 */
class SGCI_PKIMessage_CertificationRequestMessage extends
    SGCI_PKIMessage_Body
{
    /**
     * The identifier of the certificate request.
     *
     * @var integer
     */
    protected $_certReqId;

    /**
     * The template of the certificate request.
     *
     * @var SGCI_PKIMessage_CertificateTemplate
```

```
     */
    protected $_certTemplate;

    /**
     * Loads an XML
     *
     * @param DOMNode $xml The XML to be loaded
     *
     * @return void
     */
    public function loadXML (DomNode $xml)
    {
        $certReq = $xml->getElementsByTagName('certReq')->item
            (0);

        $certReqId = $certReq->getElementsByTagName('certReqId')
            ->item(0);
        $this->setCertReqId((int) $certReqId->nodeValue);

        $certTemplateXML = $certReq->getElementsByTagName('
            certTemplate')->item(0);

        $certTemplate = new SGCI_PKIMessage_CertificateTemplate
            ();
        $certTemplate->loadXML($certTemplateXML);

        $this->setCertTemplate($certTemplate);
    }

    /**
     * Sets the identifier of the certificate request
     *
     * @param int $id The id of the request
     *
     * @throws SGCI_Exception_InvalidParameter If the id is
         invalid
     * @return void
     */
    public function setCertReqId ($id)
    {
        if (is_integer($id) === true) {
            $this->_certReqId = (int) $id;
        } else {
            throw new SGCI_Exception_InvalidParameter(
                'Parameter must be an integer.' . ' ' . gettype(
                    $id) . ' ' . 'given'
            );
        }
    }

    /**
     * Gets the identifier of the certificate request
```

```
     *
     * @return number
     */
    public function getCertReqId ()
    {
        return $this->_certReqId;
    }

    /**
     * Sets the template of the certificate request
     *
     * @param SGCI_PKIMessage_CertificateTemplate $template The
          template for the request
     *
     * @return void
     */
    public function setCertTemplate (
        SGCI_PKIMessage_CertificateTemplate $template)
    {
        $this->_certTemplate = $template;
    }

    /**
     * Gets the template of the certificate request
     *
     * @return SGCI_PKIMessage_CertificateTemplate
     */
    public function getCertTemplate ()
    {
        return $this->_certTemplate;
    }

    /**
     * Returns the XML representation of the
          CertificationRequestMessage
     *
     * @param string $nodeName The name of the root node of the
         XML
     *
     * @throws SGCI_Exception if some of the required fields are
          empty
     * @return DOMElement
     */
    public function getXMLEncoded ($nodeName = 'CertReqMsg')
    {
        if ($this->_certReqId === null) {
            throw new SGCI_Exception(
                'certReqId is required and must be providaded to
                     generate XML'
            );
        }
        if ($this->_certTemplate === null) {
```

```
            throw new SGCI_Exception(
                'certTemplate is required and must be providaded
                    to generate XML'
            );
        }

        $xml = new DomDocument('1.0');

        $certReqMsg = $xml->createElement($nodeName);
        $certRequest = $xml->createElement('certReq');

        $element = $xml->createElement('certReqId', $this->
            _certReqId);

        $certRequest->appendChild($element);

        $element = $xml->importNode($this->_certTemplate->
            getXMLEncoded('certTemplate'), true);

        $certRequest->appendChild($element);
        $certReqMsg->appendChild($certRequest);

        return $certReqMsg;
    }
}
```

Código Fonte C.14: Classe SGCI_PKIMessage_CertificationRequestMessage

```
<?php
/**
 * Class that implements a PKIHeader defined in RFC4210.
 *
 * PKIHeader ::= SEQUENCE {
 * pvno                INTEGER     { cmp2000(2) },
 * sender              GeneralName,
 * recipient           GeneralName,
 * messageTime     [0] GeneralizedTime         OPTIONAL,
 * protectionAlg   [1] AlgorithmIdentifier     OPTIONAL,
 * senderKID       [2] KeyIdentifier           OPTIONAL,
 * recipKID        [3] KeyIdentifier           OPTIONAL,
 * transactionID   [4] OCTET STRING            OPTIONAL,
 * senderNonce     [5] OCTET STRING            OPTIONAL,
 * recipNonce      [6] OCTET STRING            OPTIONAL,
 * freeText        [7] PKIFreeText             OPTIONAL,
 * generalInfo     [8] SEQUENCE SIZE (1..MAX) OF
 * InfoTypeAndValue     OPTIONAL
 * }
 * PKIFreeText ::= SEQUENCE SIZE (1..MAX) OF UTF8String
 *
 */
```

```
class SGCI_PKIMessage_Header
{
    /**
     * The pvno field is fixed (at 2) for the version of the
         RFC4210 specification.
     *
     * @var numeric
     */
    protected $_pvno = 2;

    /**
     * Identifies the sender.
     *
     * @var Labsec_Security_Certification_DataTypes_GeneralName
     */
    protected $_sender;

    /**
     * Identifies the intended recipient.
     *
     * @var Labsec_Security_Certification_DataTypes_GeneralName
     */
    protected $_recipient;

    /**
     * Time of production of this message.
     *
     * Used when sender believes that the transport will be "
         suitable"; i.e.,
     * that the time will still be meaningful upon receipt
     *
     * @var Zend_Date
     */
    protected $_messageTime;

    /**
     * The algorithm used to protect the PKIMessage.
     *
     * Should only be present when the corresponding PKIMessage
         is signed.
     *
     * @var Labsec_Security_Certification_ObjectIdentifier
     */
    protected $_protectionAlg;

    /**
     * Identifies the sender key identifier.
     *
     * @var string
     */
    protected $_senderKeyId;
```

```
    /**
     * Identifies the intended recipient key identifier.
     *
     * @var string
     */
    protected $_recipientKeyId;

    /**
     * Identifies the transaction.
     *
     * I.e., this will be the same in corresponding request,
         response, certConf,
     * and PKIConf messages
     *
     * @var string
     */
    protected $_transactionId;

    /**
     * Protect the PKIMessage against replay attacks.
     *
     * @var string
     */
    protected $_senderNonce;

    /**
     * Protect the PKIMessage against replay attacks.
     *
     * @var string
     */
    protected $_recipientNounce;

    /**
     * Loads an XML
     *
     * @param DOMNode $xml The XML to be loaded
     *
     * @return void
     */
    public function loadXML (DomNode $xml)
    {
        $sender = $xml->getElementsByTagName('sender')->item(0);
        $senderObject = new
            Labsec_Security_Certification_DataTypes_DirectoryName
            ();
        $senderObject->loadXML($sender);
        $this->setSender($senderObject);
        $recipient = $xml->getElementsByTagName('recipient')->
            item(0);
        $recipientObject = new
            Labsec_Security_Certification_DataTypes_DirectoryName
            ();
```

```
        $recipientObject->loadXML($recipient);
        $this->setRecipient($recipientObject);
        $messageTime = $xml->getElementsByTagName('messageTime')
            ->item(0);
        if ($messageTime !== null) {
            $this->setMessageTime($messageTime->nodeValue);
        }
        $protectionAlg = $xml->getElementsByTagName('
            protectionAlg')->item(0);
        if ($protectionAlg !== null) {
            $oid = new
                Labsec_Security_Certification_ObjectIdentifier
                ();
            $oid->loadXML($protectionAlg);
            $this->setProtectionAlg($oid);
        }
        $senderKID = $xml->getElementsByTagName('senderKID')->
            item(0);
        if ($senderKID !== null) {
            $this->setSenderKeyId($senderKID->nodeValue);
        }
        $recipientKID = $xml->getElementsByTagName('recipKID')->
            item(0);
        if ($recipientKID !== null) {
            $this->setRecipientKeyId($recipientKID->nodeValue);
        }
        $transactionID = $xml->getElementsByTagName('
            transactionID')->item(0);
        if ($transactionID !== null) {
            $this->setTransactionId($transactionID->nodeValue);
        }
        $senderNonce = $xml->getElementsByTagName('senderNonce')
            ->item(0);
        if ($senderNonce !== null) {
            $this->setSenderNonce($senderNonce->nodeValue);
        }
        $recipientNonce = $xml->getElementsByTagName('recipNonce
            ')->item(0);
        if ($recipientNonce !== null) {
            $this->setRecipientNonce($recipientNonce->nodeValue)
                ;
        }
    }

    /**
     * Sets the sender of the message
     *
     * @param
         Labsec_Security_Certification_DataTypes_GeneralName
         $sender The sender of the message
     *
     * @return void
```

```
     */
    public function setSender
    (
        Labsec_Security_Certification_DataTypes_GeneralName
            $sender
    ) {
        $this->_sender = $sender;
    }

    /**
     * Gets the sender of the message
     *
     * @return
          Labsec_Security_Certification_DataTypes_GeneralName
     */
    public function getSender ()
    {
        return $this->_sender;
    }

    /**
     * Sets the recipient of the message
     *
     * @param
          Labsec_Security_Certification_DataTypes_GeneralName
          $recipient The recipient of the
     *

          message
     *
     * @return void
     */
    public function setRecipient
    (
        Labsec_Security_Certification_DataTypes_GeneralName
            $recipient
    ) {
        $this->_recipient = $recipient;
    }

    /**
     * Gets the recipient of the message
     *
     * @return
          Labsec_Security_Certification_DataTypes_GeneralName
     */
    public function getRecipient ()
    {
        return $this->_recipient;
    }

    /**
```

```
     * Sets the time of production of this message
     *
     * @param string|integer|Zend_Date|array $time The time of
         production of the message
     *
     * @throws Zend_Date_Exception If the provided time is
         invalid
     * @return void
     */
    public function setMessageTime ($time)
    {
        $this->_messageTime = new Zend_Date($time);
    }

    /**
     * Gets the time of production of this message
     *
     * @return Zend_Date
     */
    public function getMessageTime ()
    {
        return $this->_messageTime;
    }

    /**
     * Sets the protection algorithm used to protect the
         PKIMessage
     *
     * @param Labsec_Security_Certification_ObjectIdentifier
         $oid The OID of the algorithm used to
     *

         sign the PKIMessage
     *
     * @return void
     */
    public function setProtectionAlg (
        Labsec_Security_Certification_ObjectIdentifier $oid)
    {
        $this->_protectionAlg = $oid;
    }

    /**
     * Gets the protection algorithm used to protect the
         PKIMessage
     *
     * @return Labsec_Security_Certification_ObjectIdentifier
     */
    public function getProtectionAlg ()
    {
        return $this->_protectionAlg;
    }
```

```
    /**
     * Sets the sender key identifier
     *
     * @param string $senderKeyId The key identifier of the
         sender public key
     *
     * @throws SGCI_Exception_InvalidParameter If the
         $senderKeyId is invalid
     * @return void
     */
    public function setSenderKeyId ($senderKeyId)
    {
        if (is_string($senderKeyId) === true) {
            $this->_senderKeyId = $senderKeyId;
        } else {
            throw new SGCI_Exception_InvalidParameter('The key
                identifier must be a string');
        }
    }

    /**
     * Gets the sender key identifier
     *
     * @return string
     */
    public function getSenderKeyId ()
    {
        return $this->_senderKeyId;
    }

    /**
     * Sets the recipient key identifier
     *
     * @param string $recipientKeyId The key identifier of the
         recipient public key
     *
     * @throws SGCI_Exception_InvalidParameter If the
         $recipientKeyId is invalid
     * @return void
     */
    public function setRecipientKeyId ($recipientKeyId)
    {
        if (is_string($recipientKeyId) === true) {
            $this->_recipientKeyId = $recipientKeyId;
        } else {
            throw new SGCI_Exception_InvalidParameter('The key
                identifier must be a string');
        }
    }

    /**
```

```
     * Gets the recipient key identifier
     *
     * @return string
     */
    public function getRecipientKeyId ()
    {
        return $this->_recipientKeyId;
    }

    /**
     * Sets the identifier of the transaction
     *
     * @param string|integer $id The transaction identifier
     *
     * @throws SGCI_Exception_InvalidParameter If the $id is
         invalid
     * @return void
     */
    public function setTransactionId ($id)
    {
        if ((is_string($id) === true)
            || (is_integer($id) === true)
        ) {
            $this->_transactionId = (string) $id;
        } else {
            throw new SGCI_Exception_InvalidParameter(
                'Parameter must be a string.' . ' ' . gettype(
                    $id) . ' ' . 'given'
            );
        }
    }

    /**
     * Gets the identifier of the transaction
     *
     * @return string
     */
    public function getTransactionId ()
    {
        return $this->_transactionId;
    }

    /**
     * Sets the recipient nonce of this message
     *
     * @param string|integer $nonce The nonce of the recipient
         of this message
     *
     * @throws SGCI_Exception_InvalidParameter If the providaded
          parameter is invalid
     * @return void
     */
```

```
    public function setRecipientNonce ($nonce)
    {
        if ((is_string($nonce) === true)
            || (is_integer($nonce) === true)
        ) {
            $this->_recipientNounce = (string) $nonce;
        } else {
            throw new SGCI_Exception_InvalidParameter(
                'Parameter must be a string.' . ' ' . gettype(
                    $nonce) . ' ' . 'given'
            );
        }
    }

    /**
     * Gets the recipient nonce of this message
     *
     * @return string
     */
    public function getRecipientNonce ()
    {
        return $this->_recipientNounce;
    }

    /**
     * Sets the sender nonce of this message
     *
     * @param string|integer $nonce The nonce of the sender of
         this message
     *
     * @throws SGCI_Exception_InvalidParameter If the providaded
          parameter is invalid
     * @return void
     */
    public function setSenderNonce ($nonce)
    {
        if ((is_string($nonce) === true) || (is_integer($nonce)
            === true)) {
            $this->_senderNonce = (string) $nonce;
        } else {
            throw new SGCI_Exception_InvalidParameter(
                'Parameter must be a string.' . ' ' . gettype(
                    $nonce) . ' ' . 'given'
            );
        }
    }

    /**
     * Gets the sender nonce of this message
     *
     * @return string
     */
```

```
public function getSenderNonce ()
{
    return $this->_senderNonce;
}

/**
 * Returns the XML representation of the PKIHeader
 *
 * @param string $nodeName The name of the root node of the
     XML
 *
 * @throws SGCI_Exception If some of the required fields are
      empty
 * @return DOMElement
 */
public function getXMLEncoded ($nodeName = 'PKIHeader')
{
    if ($this->_sender === null) {
        throw new SGCI_Exception('sender is required and
            must be providaded to generate XML');
    }
    if ($this->_recipient === null) {
        throw new SGCI_Exception(
            'recipient is required and must be providaded to
                 generate XML'
        );
    }

    $xml = new DomDocument('1.0');

    $node = $xml->createElement($nodeName);
    $element = $xml->createElement('pvno', $this->_pvno);
    $node->appendChild($element);

    $element = $xml->importNode($this->_sender->
        getXMLEncoded('sender'), true);
    $node->appendChild($element);

    $element = $xml->importNode($this->_recipient->
        getXMLEncoded('recipient'), true);
    $node->appendChild($element);

    $node = $this->_appendOptionalFields($node);

    return $node;
}

protected function _appendOptionalFields (DOMNode $node)
{
    $xml = new DOMDocument();
    $node = $xml->importNode($node, true);
```

```
        if ($this->_messageTime !== null) {
            $element = $xml->createElement('messageTime', $this
                ->_messageTime->getTimestamp());
            $node->appendChild($element);
        }
        if ($this->_protectionAlg !== null) {
            $element = $this->_protectionAlg->getXMLEncoded('
                protectionAlg');
            $element = $xml->importNode($element, true);
            $node->appendChild($element);
        }
        if ($this->_senderKeyId !== null) {
            $element = $xml->createElement('senderKID', $this->
                _senderKeyId);
            $node->appendChild($element);
        }
        if ($this->_recipientKeyId !== null) {
            $element = $xml->createElement('recipKID', $this->
                _recipientKeyId);
            $node->appendChild($element);
        }
        if ($this->_transactionId !== null) {
            $element = $xml->createElement('transactionID',
                $this->_transactionId);
            $node->appendChild($element);
        }
        if ($this->_senderNonce !== null) {
            $element = $xml->createElement('senderNonce', $this
                ->_senderNonce);
            $node->appendChild($element);
        }
        if ($this->_recipientNounce !== null) {
            $element = $xml->createElement('recipNonce', $this->
                _recipientNounce);
            $node->appendChild($element);
        }
        return $node;
    }
}
```

Código Fonte C.15: Classe SGCI_PKIMessage_Header

```
<?php
class SGCI_PKIMessage_InvalidMessageException extends
    SGCI_Exception
{

}
```

Código Fonte C.16: Classe SGCI_PKIMessage_InvalidMessageException

```php
<?php
/**
 * Class that implements a cr defined as a choice of the PKIBody
      defined in RFC4210.
 *
 * cr        [2]   CertReqMessages,        --Certification Req
 *
 */
class SGCI_PKIMessage_Body_CertificationRequestMessage
    extends SGCI_PKIMessage_Body_CertificationRequestMessages
{
    /**
     * Returns the XML representation of the CertReqMessages
     *
     * @param string $nodeName The name of the root node of the
          XML
     *
     * @return DOMElement
     */
    public function getXMLEncoded ($nodeName = 'cr')
    {
        return parent::getXMLEncoded($nodeName);
    }
}
```

Código Fonte C.17: Classe SGCI_PKIMessage_Body_CertificationRequestMessage

```php
<?php
/**
 * Class that implements a CertReqMessages defined in RFC2511.
 *
 * CertReqMessages ::= SEQUENCE SIZE (1..MAX) OF CertReqMsg
 *
 */
class SGCI_PKIMessage_Body_CertificationRequestMessages extends
    SGCI_PKIMessage_Body
{
    /**
     * The identifier of the certificate request.
     *
     * @var array of SGCI_PKIMessage_CertificationRequestMessage
     */
    protected $_certReqMsgs = array();

    /**
     * Loads an XML
     *
     * @param DOMNode $xml The XML to be loaded
     *
     * @return void
```

```
     */
    public function loadXML (DomNode $xml)
    {
        $certReqMsgs = $xml->getElementsByTagName('CertReqMsg');
        foreach ($certReqMsgs as $certReqMsgXML) {
            $certReqMsg = new
                SGCI_PKIMessage_CertificationRequestMessage();
            $certReqMsg->loadXML($certReqMsgXML);
            $this->addCertReqMsg($certReqMsg);
        }
    }

    /**
     * Adds a certificate request to the list of requests
     *
     * @param SGCI_PKIMessage_CertificationRequestMessage
         $certReqMsg The certificate request
     *

         to be added
     *
     * @return void
     */
    public function addCertReqMsg
    (
        SGCI_PKIMessage_CertificationRequestMessage $certReqMsg
    ) {
        $this->_certReqMsgs[] = $certReqMsg;
    }

    /**
     * Gets all the certificate requests
     *
     * @return array of
         SGCI_PKIMessage_CertificationRequestMessage
     */
    public function getCertReqMsgs ()
    {
        return $this->_certReqMsgs;
    }

    /**
     * Returns the XML representation of the CertReqMessages
     *
     * @param string $nodeName The name of the root node of the
         XML
     *
     * @throws SGCI_Exception if the CertReqMessages is empty
     * @return DOMElement
     */
    public function getXMLEncoded ($nodeName = 'CertReqMessages'
        )
```

```
    {
        if (empty($this->_certReqMsgs) === true) {
            throw new SGCI_Exception(
                'CertReqMessages is required and must have at
                    least one entry to generate XML'
            );
        }

        $xml = new DomDocument('1.0');

        $node = $xml->createElement($nodeName);
        foreach ($this->_certReqMsgs as $certReqMsg) {
            $element = $certReqMsg->getXMLEncoded();
            $element = $xml->importNode($element, true);
            $node->appendChild($element);
        }
        return $node;
    }
}
```

Código Fonte C.18: Classe SGCI_PKIMessage_Body_CertificationRequestMessages

```
<?php
/**
 * Class that implements a CertRepMessage defined in RFC4210.
 *
 * CertRepMessage ::= SEQUENCE {
 * caPubs           [1] SEQUENCE SIZE (1..MAX) OF Certificate
     OPTIONAL,
 * response             SEQUENCE OF CertResponse
 * }
 *
 * CertResponse ::= SEQUENCE {
 * certReqId            INTEGER,
 * status               PKIStatusInfo,
 * certifiedKeyPair     CertifiedKeyPair     OPTIONAL,
 * rspInfo              OCTET STRING         OPTIONAL
 * -- analogous to the id-regInfo-utf8Pairs string defined
 * -- for regInfo in CertReqMsg [CRMF]
 * }
 *
 * CertifiedKeyPair ::= SEQUENCE {
 * certOrEncCert        CertOrEncCert,
 * privateKey       [0] EncryptedValue       OPTIONAL,
 * -- see [CRMF] for comment on encoding
 * publicationInfo [1] PKIPublicationInfo   OPTIONAL
 * }
 *
 * CertOrEncCert ::= CHOICE {
 * certificate      [0] Certificate,
```

```
 * encryptedCert    [1] EncryptedValue
 * }
 *
 */
class SGCI_PKIMessage_Body_CertificationResponse extends
    SGCI_PKIMessage_Body
{
    /**
     * Aditional certificates.
     *
     * @var array of
          LabsecCL_Security_Certification_CertificateCL
     */
    protected $_caPubs = array();

    /**
     * The id of the certificate request.
     *
     * @var integer
     */
    protected $_certReqId;

    /**
     * The status of the requested certificate.
     *
     * @var SGCI_PKIStatusInfo
     */
    protected $_status;

    /**
     * The certificate itself.
     *
     * It will be present only if status is:
     * accepted                (0), OR
     * grantedWithMods         (1),
     *
     * @var LabsecCL_Security_Certification_CertificateCL
     */
    protected $_certificate;

    /**
     * Loads an XML
     *
     * @param DOMNode $xml The XML to be loaded
     *
     * @return void
     */
    public function loadXML (DomNode $xml)
    {
        $caPubs = $xml->getElementsByTagName('caPubs')->item(0);
        if ($caPubs !== null) {
            $certificates = $caPubs->getElementsByTagName('
```

```
            certificate');
        foreach ($certificates as $certificate) {
            $certificateObject = new
                LabsecCL_Security_Certification_CertificateCL
                (
                $certificate->nodeValue
            );
            $this->addCaPub($certificateObject);
        }
    }

    $response = $xml->getElementsByTagName('response')->item
        (0);
    $certReqId = $response->getElementsByTagName('certReqId'
        )->item(0);
    $this->setCertReqId((int) $certReqId->nodeValue);

    $status = $response->getElementsByTagName('status')->
        item(0);
    $statusInfo = new SGCI_PKIStatusInfo();
    $statusInfo->loadXML($status);
    $this->setStatus($statusInfo);

    $certifiedKeyPair = $response->getElementsByTagName('
        certifiedKeyPair')->item(0);
    if ($certifiedKeyPair !== null) {
        $certOrEncCert = $certifiedKeyPair->
            getElementsByTagName('certOrEncCert')->item(0);
        $certificate = $certOrEncCert->getElementsByTagName(
            'certificate')->item(0);
        $certificateObject = new
            LabsecCL_Security_Certification_CertificateCL(
            $certificate->nodeValue
        );
        $this->setCertificate($certificateObject);
    }
}

/**
 * Adds a certificate to the caPubs field
 *
 * @param Labsec_Security_Certification_Certificate
     $certificate An additional certificate
 *
 * @return void
 */
public function addCaPub
(
    Labsec_Security_Certification_Certificate $certificate
) {
    $this->_caPubs[] = $certificate;
}
```

```
    /**
     * Gets all certificates in the caPubs field
     *
     * @return array of
         Labsec_Security_Certification_Certificate
     */
    public function getCaPubs ()
    {
        return $this->_caPubs;
    }

    /**
     * Sets the certReqId field
     *
     * @param integer $certReqId The id of the request
     *
     * @throws SGCI_Exception_InvalidParameter If the $certReqId
         is invalid
     * @return void
     */
    public function setCertReqId ($certReqId)
    {
        if (is_integer($certReqId) === true) {
            $this->_certReqId = $certReqId;
        } else {
            throw new SGCI_Exception_InvalidParameter(
                'certReqId must be an integer.' . ' ' . gettype(
                    $certReqId) . ' ' . 'provided'
            );
        }
    }

    /**
     * Gets the certReqId field
     *
     * @return integer
     */
    public function getCertReqId ()
    {
        return $this->_certReqId;
    }

    /**
     * Sets the status field
     *
     * @param SGCI_PKIStatusInfo $status The status of the
         response for the request
     *
     * @throws SGCI_Exception_InvalidParameter If the $status is
         not valid
     * @return void
```

```
     */
    public function setStatus (SGCI_PKIStatusInfo $status)
    {
        if ($status->getStatus() !== null) {
            $this->_status = $status;
        } else {
            throw new SGCI_Exception_InvalidParameter('status
                must not be empty');
        }
    }

    /**
     * Gets the status field
     *
     * @return SGCI_PKIStatusInfo
     */
    public function getStatus ()
    {
        return $this->_status;
    }

    /**
     * Sets the certificate field
     *
     * @param Labsec_Security_Certification_Certificate
         $certificate The certificate issued for
     *

         the request
     *
     * @throws SGCI_Exception If the status was not yet set or
         is not
     *                        ACCEPTED or GRANTED WITH MODS
         status
     * @return void
     *
     */
    public function setCertificate
    (
        Labsec_Security_Certification_Certificate $certificate
    ) {
        if ($this->_status !== null) {
            $status = $this->_status->getStatus();
        } else {
            throw new SGCI_Exception(
                'Certificate cannot be setted. Status is not
                    setted.'
            );
        }

        if ($status === SGCI_PKIStatus::STATUS_ACCEPTED
            || $status === SGCI_PKIStatus::
```

```
                STATUS_GRANTED_WITH_MODS
        ) {
            $this->_certificate = $certificate;
        } else {
            throw new SGCI_Exception(
                'Certificate cannot be setted. status should be
                    ACCEPTED or GRANTED WITH MODS'
            );
        }
    }

    /**
     * Gets the certificate field
     *
     * @return Labsec_Security_Certification_Certificate
     */
    public function getCertificate ()
    {
        return $this->_certificate;
    }

    /**
     * Returns the XML representation of the CertRepMessage
     *
     * @param string $nodeName The name of the root node of the
         XML
     *
     * @throws SGCI_Exception If some of the required fields are
          empty
     * @return DOMElement
     */
    public function getXMLEncoded ($nodeName = 'CertRepMessage')
    {
        if ($this->_certReqId === null) {
            throw new SGCI_Exception(
                'certReqId is required and must be provided to
                    generate XML'
            );
        }
        if ($this->_status === null) {
            throw new SGCI_Exception(
                'status is required and must be provided to
                    generate XML'
            );
        }

        $xml = new DomDocument('1.0');

        $node = $xml->createElement($nodeName);
        if (empty($this->_caPubs) === false) {
            $caPubs = $xml->createElement('caPubs');
            foreach ($this->_caPubs as $caPub) {
```

```
                $element = $xml->createElement('certificate',
                    $caPub->getPem());
                $caPubs->appendChild($element);
            }
            $node->appendChild($caPubs);
        }
        $response = $xml->createElement('response');
        $element = $xml->createElement('certReqId', $this->
            _certReqId);
        $response->appendChild($element);

        $element = $this->_status->getXMLEncoded('status');
        $element = $xml->importNode($element, true);
        $response->appendChild($element);
        if ($this->_certificate !== null) {
            $certifiedKeyPair = $xml->createElement('
                certifiedKeyPair');
            $certOrEncCert = $xml->createElement('certOrEncCert'
                );
            $element = $xml->createElement('certificate', $this
                ->_certificate->getPem());
            $certOrEncCert->appendChild($element);
            $certifiedKeyPair->appendChild($certOrEncCert);
            $response->appendChild($certifiedKeyPair);
        }
        $node->appendChild($response);
        return $node;
    }
}
```

Código Fonte C.19: Classe SGCI_PKIMessage_Body_CertificationResponse

```
<?php
/**
 * Class that implements a cp defined as a choice of the PKIBody
      defined in RFC4210.
 *
 * cp        [3]   CertRepMessage,          --Certification
     Response
 *
 */
class SGCI_PKIMessage_Body_CertificationResponseMessage
    extends SGCI_PKIMessage_Body_CertificationResponse
{
    /**
     * Returns the XML representation of the CertRepMessage
     *
     * @param string $nodeName The name of the root node of the
         XML
     *
```

```
     * @return DOMElement
     */
    public function getXMLEncoded ($nodeName = 'cp')
    {
        return parent::getXMLEncoded($nodeName);
    }
}
```

Código Fonte C.20: Classe SGCI_PKIMessage_Body_CertificationResponseMessage

```
<?php
/**
 * Class that implements a ir defined as a choice of the PKIBody
      defined in RFC4210.
 *
 * ir       [0]  CertReqMessages,        --Initialization Req
 *
 */
class SGCI_PKIMessage_Body_InitializationRequestMessage
    extends SGCI_PKIMessage_Body_CertificationRequestMessages
{
    /**
     * Returns the XML representation of the CertReqMessages
     *
     * @param string $nodeName The name of the root node of the
          XML
     *
     * @throws SGCI_Exception if the CertReqMessages is empty
     * @return DOMElement
     */
    public function getXMLEncoded ($nodeName = 'ir')
    {
        return parent::getXMLEncoded($nodeName);
    }
}
```

Código Fonte C.21: Classe SGCI_PKIMessage_Body_InitializationRequestMessage

```
<?php
/**
 * Class that implements a ip defined as a choice of the PKIBody
      defined in RFC4210.
 *
 * ip       [1]  CertRepMessage,         --Initialization
     Response
 *
 */
class SGCI_PKIMessage_Body_InitializationResponseMessage
    extends SGCI_PKIMessage_Body_CertificationResponse
```

```
{
    /**
     * Returns the XML representation of the CertRepMessage
     *
     * @param string $nodeName The name of the root node of the
         XML
     *
     * @throws SGCI_Exception If some of the required fields are
         empty
     * @return DOMElement
     */
    public function getXMLEncoded ($nodeName = 'ip')
    {
        return parent::getXMLEncoded($nodeName);
    }
}
```

Código Fonte C.22: Classe SGCI_PKIMessage_Body_InitializationResponseMessage

```
<?php
/**
 * Class that implements a request for listing the certificates
     of the
 * registered entities in an instance of the SGCI.
 *
 * This class should be used as a new choice option of the
     PKIBody structure,
 * defined in RFC4210.
 *
 * ListCertsReq ::= NULL
 *
 */
class SGCI_PKIMessage_Body_ListCertificatesRequest
    extends SGCI_PKIMessage_Body
{
   /**
    * Loads an XML.
    *
    * @param DOMNode $xml The XML to be loaded
    *
    * @return void
    */
    public function loadXML(DOMNode $xml)
    {
        //Nothing to do, since this message is a null value
    }

    /**
     * Returns the XML representation of the PKIBody
     *
```

```
     * @param string $nodeName The name of the root node of the
         XML
     *
     * @return DOMElement
     */
    public function getXMLEncoded ($nodeName = 'listCertReq')
    {
        $xml = new DOMDocument();

        $node = $xml->createElement($nodeName);

        //nothing to add, since this message has a NULL value

        return $node;
    }
}
```

Código Fonte C.23: Classe SGCI_PKIMessage_Body_ListCertificatesRequest

```
<?php
/**
 * Class that implements the response for the request for
     listing the certificates of the
 * registered entities in an instance of the SGCI.
 *
 * This class should be used as a new choice option of the
     PKIBody structure,
 * defined in RFC4210.
 *
 * ListCertsReq ::= SEQUENCE OF Certificate
 *
 */
class SGCI_PKIMessage_Body_ListCertificatesResponse
    extends SGCI_PKIMessage_Body
{

    /**
     * The certificates that matches with the request.
     *
     * @var array
     */
    protected $_certificates = array();

   /**
    * Loads an XML.
    *
    * The XML should be a the genp choice of the PKIBody
    *
    * @param DOMNode $xml The XML to be loaded
    *
```

```
* @throws SGCI_Exception If the OID in the XML does not
    correspond to the OID of this structure
* @return void
*/
public function loadXML(DOMNode $xml)
{
    $certificatesXml = $xml->getElementsByTagName('
        certificate');

    foreach ($certificatesXml as $certificateXml) {
        $data = $certificateXml->nodeValue;
        $certificate = new
            LabsecCL_Security_Certification_CertificateCL(
            $data);
        $this->addCertificate($certificate);
    }
}

/**
 * Adds a certificate to the response
 *
 * @param Labsec_Security_Certification_Certificate
     $certificate The certificate to be added
 *

     to the response
 *
 * @return void
 */
public function addCertificate(
    Labsec_Security_Certification_Certificate $certificate)
{
    $this->_certificates[] = $certificate;
}

/**
 * Adds various certificates to the response
 *
 * @param array $certificates The certificates to be added
     to the response
 *
 * @return void
 */
public function addCertificates(array $certificates)
{
    foreach ($certificates as $certificate) {
        $this->addCertificate($certificate);
    }
}

/**
 * Returns all the added certificates
```

```
     *
     * @return array of
          Labsec_Security_Certification_Certificate
     */
    public function getCertificates()
    {
        return $this->_certificates;
    }

    /**
     * Returns the XML representation of the ListCertsRep
     *
     * @param string $nodeName The name of the root node of the
          XML
     *
     * @return DOMElement
     */
    public function getXMLEncoded ($nodeName = 'listCertRep')
    {
        $xml = new DOMDocument();

        $value = $xml->createElement($nodeName);

        foreach ($this->_certificates as $certificate) {
            $element = $xml->createElement('certificate',
                $certificate->getPem());
            $value->appendChild($element);
        }

        return $value;
    }
}
```

Código Fonte C.24: Classe SGCI_PKIMessage_Body_ListCertificatesResponse

```
<?php
/**
 * Class that implements a PollReqContent defined in RFC4210.
 *
 * PollReqContent ::= SEQUENCE OF SEQUENCE {
 * certReqId              INTEGER
 * }
 */
class SGCI_PKIMessage_Body_PollingRequestContent extends
    SGCI_PKIMessage_Body
{
    /**
     * Ids for the requests.
     *
     * @var array of integer
```

```
     */
    protected $_certReqIds = array();

    /**
     * Loads an XML
     *
     * @param DOMNode $xml The XML to be loaded
     *
     * @return void
     */
    public function loadXML (DOMNode $xml)
    {
        $certReqIds = $xml->getElementsByTagName('certReqId');

        foreach ($certReqIds as $certReqId) {
            $this->addCertReqId((int) $certReqId->nodeValue);
        }
    }

    /**
     * Adds a certReqId
     *
     * @param integer $id The identifier of the request
     *
     * @throws SGCI_Exception_InvalidParameter If the id is not
         an integer
     * @return void
     */
    public function addCertReqId ($id)
    {
        if (is_integer($id) === true) {
            $this->_certReqIds[] = $id;
        } else {
            throw new SGCI_Exception_InvalidParameter(
                'Parameter must be an integer.' . ' ' . gettype(
                    $id) . ' ' . 'given'
            );
        }
    }

    /**
     * Returns all the certReqIds added
     *
     * @return array of integer
     */
    public function getCertReqids ()
    {
        return $this->_certReqIds;
    }

    /**
     * Returns the XML representation of the PollReqContent
```

```
     *
     * @param string $nodeName The name of the root node of the
       XML
     *
     * @throws SGCI_Exception if the CertReqIds is empty
     * @return DOMElement
     */
    public function getXMLEncoded ($nodeName = 'PollReqContent')
    {
        if (empty($this->_certReqIds) === true) {
            throw new SGCI_Exception('At least one certReqId
                must be provided to generate XML');
        }

        $xml = new DomDocument('1.0');

        $pollReqContent = $xml->createElement($nodeName);
        foreach ($this->_certReqIds as $certReqId) {
            $element = $xml->createElement('certReqId',
                $certReqId);
            $pollReqContent->appendChild($element);
        }

        return $pollReqContent;
    }
}
```

Código Fonte C.25: Classe SGCI_PKIMessage_Body_PollingRequestContent

```
<?php
/**
 * Class that implements a pollReq defined as a choice of the
     PKIBody defined in RFC4210.
 *
 * pollReq  [25] PollReqContent,          --Polling request
 *
 */
class SGCI_PKIMessage_Body_PollingRequestMessage
    extends SGCI_PKIMessage_Body_PollingRequestContent
{
    /**
     * Returns the XML representation of the PollReqContent
     *
     * @param string $nodeName The name of the root node of the
       XML
     *
     * @throws SGCI_Exception if the CertReqIds is empty
     * @return DOMElement
     */
    public function getXMLEncoded ($nodeName = 'pollReq')
```

```
    {
        return parent::getXMLEncoded($nodeName);
    }
}
```

Código Fonte C.26: Classe SGCI_PKIMessage_Body_PollingRequestMessage

```
<?php
/**
 * Class that implements a PollRepContent defined in RFC4210.
 *
 * PollRepContent ::= SEQUENCE OF SEQUENCE {
 * certReqId     INTEGER,
 * checkAfter    INTEGER,  -- time in seconds
 * reason        PKIFreeText OPTIONAL }
 *
 */
class SGCI_PKIMessage_Body_PollingResponseContent extends
    SGCI_PKIMessage_Body
{

    /**
     * Array containing the polling responses.
     *
     * This array should be like:
     *
     * pollReq[]['certReqId'] = integer
     * pollReq[]['checkAfter'] = integer
     * pollReq[ ]['reason'] = string
     *
     * @var array
     */
    protected $_pollRep = array();

    const CERT_REQ_ID = 'certReqId';
    const CHECK_AFTER = 'checkAfter';
    const REASON = 'reason';

    /**
     * Loads an XML
     *
     * @param DOMNode $xml The XML to be loaded
     *
     * @return void
     */
    public function loadXML (DOMNode $xml)
    {
        $sequences = $xml->getElementsByTagName('sequence');

        foreach ($sequences as $sequence) {
```

```
            $certReqId = $sequence->getElementsByTagName(self::
                CERT_REQ_ID)->item(0)->nodeValue;
            $checkAfter = $sequence->getElementsByTagName(self::
                CHECK_AFTER)->item(0)->nodeValue;

            $reason = null;
            $reasonXML = $sequence->getElementsByTagName(self::
                REASON)->item(0);
            if ($reasonXML !== null) {
                $reason = $reasonXML->nodeValue;
            }
            $this->addPollingResponse((int) $certReqId, (int)
                $checkAfter, $reason);
        }
    }

    /**
     * Adds a polling response
     *
     * @param integer $certReqId  The request id
     * @param integer $checkAfter Time for the next check of the
          status of the request
     * @param string  $reason     The reason why the request was
          not yet processed
     *
     * @throws SGCI_Exception_InvalidParameter If some of the
          parameters are incorrect
     * @return void
     */
    public function addPollingResponse ($certReqId, $checkAfter,
         $reason = null)
    {
        if (is_int($certReqId) === false) {
            throw new SGCI_Exception_InvalidParameter(
                'Parameter must be an integer.' . ' ' . gettype(
                    $certReqId) . ' ' . 'given'
            );
        }
        if (is_int($checkAfter) === false) {
            throw new SGCI_Exception_InvalidParameter(
                'Parameter must be an integer.' . ' ' . gettype(
                    $checkAfter) . ' ' . 'given'
            );
        }
        if (($reason !== null) && (is_string($reason) === false)
            ) {
            throw new SGCI_Exception_InvalidParameter(
                'Parameter must be a string.' . ' ' . gettype(
                    $reason) . ' ' . 'given'
            );
        }
```

```
        $response = array();
        $response[self::CERT_REQ_ID] = $certReqId;
        $response[self::CHECK_AFTER] = $checkAfter;
        $response[self::REASON] = $reason;

        $this->_pollRep[] = $response;
    }

    /**
     * Returns all the polling responses
     *
     * @return array
     */
    public function getPollingResponses ()
    {
        return $this->_pollRep;
    }

    /**
     * Returns the XML representation of the PollRepContent
     *
     * @param string $nodeName The name of the root node of the
          XML
     *
     * @throws SGCI_Exception if no polling response was added
     * @return DOMElement
     */
    public function getXMLEncoded ($nodeName = 'PollRepContent')
    {
        if (empty($this->_pollRep) === true) {
            throw new SGCI_Exception(
                'At least one polling response must be provided
                    to generate XML'
            );
        }

        $xml = new DomDocument('1.0');
        $pollRepContent = $xml->createElement($nodeName);

        foreach ($this->_pollRep as $response) {
            $sequence = $xml->createElement('sequence');

            $certReqId = $xml->createElement('certReqId',
                $response[self::CERT_REQ_ID]);
            $sequence->appendChild($certReqId);

            $checkAfter = $xml->createElement('checkAfter',
                $response[self::CHECK_AFTER]);
            $sequence->appendChild($checkAfter);

            $reason = $response[self::REASON];
            if (($reason) !== null) {
```

```
                $reasonXML = $xml->createElement('reason',
                    $reason);
                $sequence->appendChild($reasonXML);
            }
            $pollRepContent->appendChild($sequence);
        }
        return $pollRepContent;
    }
}
```

Código Fonte C.27: Classe SGCI_PKIMessage_Body_PollingResponseContent

```
<?php
/**
 * Class that implements a pollRep defined as a choice of the
     PKIBody defined in RFC4210.
 *
 * pollRep  [26] PollRepContent,          --Polling response
 *
 */
class SGCI_PKIMessage_Body_PollingResponseMessage
    extends SGCI_PKIMessage_Body_PollingResponseContent
{
    /**
     * Returns the XML representation of the PollRepContent
     *
     * @param string $nodeName The name of the root node of the
         XML
     *
     * @return DOMElement
     */
    public function getXMLEncoded ($nodeName = 'pollRep')
    {
        return parent::getXMLEncoded($nodeName);
    }
}
```

Código Fonte C.28: Classe SGCI_PKIMessage_Body_PollingResponseMessage

```
<?php
/**
 * Class that implements a RevReqContent defined in RFC4210.
 *
 * RevReqContent ::= SEQUENCE OF RevDetails
 *
 * RevDetails ::= SEQUENCE {
 * certDetails          CertTemplate,
 * -- allows requester to specify as much as they can about
 * -- the cert. for which revocation is requested
```

```
 * -- (e.g., for cases in which serialNumber is not available)
 * crlEntryDetails      Extensions        OPTIONAL -- requested
     crlEntryExtensions
 * }
 *
 */
class SGCI_PKIMessage_Body_RevocationRequestContent extends
    SGCI_PKIMessage_Body
{
    /**
     * Information about the certificates for wich revocation is
         requested.
     *
     * The array should be like:
     *
     * revDetails[][''certTemplate''] = instance of
         SGCI_PKIMessage_CertificateTemplate
     * revDetails[][''crlEntryDetails''] = array of
         Labsec_Security_Certification_Extension
     *
     * @var array
     */
    protected $_revDetails = array();

    const CERT_TEMPLATE = 'certTemplate';
    const CRL_ENTRY_DETAILS = 'crlEntryDetails';

    /**
     * Loads an XML
     *
     * @param DOMNode $xml The XML to be loaded
     *
     * @return void
     */
    public function loadXML (DomNode $xml)
    {
        $revDetails = $xml->getElementsByTagName('RevDetails');

        foreach ($revDetails as $revDetail) {
            $template = new SGCI_PKIMessage_CertificateTemplate
                ();
            $templateXML = $revDetail->getElementsByTagName('
                certDetails')->item(0);
            $template->loadXML($templateXML);

            $crlEntryDetails = array();
            $crlEntryDetailsXML = $revDetail->
                getElementsByTagName('crlEntryDetails');

            foreach ($crlEntryDetailsXML as $extension) {
                $extensionXML = $extension->getElementsByTagName
                    ('Extension')->item(0);
```

```
                $extensionObject =
                    Labsec_Security_Certification_ExtensionFactory
                        ::getExtensionByXML(
                        $extensionXML
                    );

                $crlEntryDetails[] = $extensionObject;
            }
            $this->addRevocationRequest($template,
                $crlEntryDetails);
        }
    }

    /**
     * Adds a certificate revocation request to the message
     *
     * @param SGCI_PKIMessage_CertificateTemplate $certTemplate
           Information about the
     *

          certificate for wich
     *

          revocation is requested
     * @param array
          $crlEntryDetails Requested crlEntryExtensions
     *
     * @throws SGCI_Exception_InvalidParameter If some of the
          parameters aew invalid
     * @return void
     */
    public function addRevocationRequest
    (
        SGCI_PKIMessage_CertificateTemplate $certTemplate,
        array $crlEntryDetails = array()
    ) {
        $arrayTemp = array();
        if ($certTemplate->isEmpty() === false) {
            $arrayTemp[self::CERT_TEMPLATE] = $certTemplate;
        } else {
            throw new SGCI_Exception_InvalidParameter(
                'At least one field of certTemplate must be
                    present'
            );
        }
        foreach ($crlEntryDetails as $crlEntryDetail) {
            if ($crlEntryDetail instanceof
                Labsec_Security_Certification_Extension) {
                $arrayTemp[self::CRL_ENTRY_DETAILS][] =
                    $crlEntryDetail;
            } else {
                throw new SGCI_Exception_InvalidParameter(
```

```
                'crlEntryDetail must be an instance of' . '
                    '
                . 'Labsec_Security_Certification_Extension.'
                    . ' '
                . gettype($crlEntryDetail) . ' ' . 'provided
                    '
            );
        }
    }
    $this->_revDetails[] = $arrayTemp;
}

/**
 * Returns all certTemplates added
 *
 * @return array of SGCI_PKIMessage_CertificateTemplate
 */
public function getRevocationRequests ()
{
    return $this->_revDetails;
}

/**
 * Returns the XML representation of the RevReqContent
 *
 * @param string $nodeName The name of the root node of the
     XML
 *
 * @throws SGCI_Exception If the RevReqContent is empty
 * @return DOMElement
 */
public function getXMLEncoded ($nodeName = 'RevReqContent')
{
    if (empty($this->_revDetails) === true) {
        throw new SGCI_Exception(
            'Must be at least one revocation request to
                generate XML'
        );
    }

    $xml = new DomDocument('1.0');

    $node = $xml->createElement($nodeName);
    foreach ($this->_revDetails as $revDetail) {
        $revDetailXML = $xml->createElement('RevDetails');
        $certDetails = $revDetail[self::CERT_TEMPLATE]->
            getXMLEncoded('certDetails');
        $certDetails = $xml->importNode($certDetails, true);
        $revDetailXML->appendChild($certDetails);
        if (array_key_exists(self::CRL_ENTRY_DETAILS,
            $revDetail) === true) {
            $crlEntryDetails = $revDetail[self::
```

```
                    CRL_ENTRY_DETAILS];
                $extensions = $xml->createElement('
                    crlEntryDetails');
                foreach ($crlEntryDetails as $crlEntryDetail) {
                    $extension = $crlEntryDetail->getXMLEncoded
                        ();
                    $extension = $xml->importNode($extension,
                        true);
                    $extensions->appendChild($extension);
                }
                $revDetailXML->appendChild($extensions);
            }
            $node->appendChild($revDetailXML);
        }
        return $node;
    }
}
```

Código Fonte C.29: Classe SGCI_PKIMessage_Body_RevocationRequestContent

```
<?php
/**
 * Class that implements a rr defined as a choice of the PKIBody
      defined in RFC4210.
 *
 * rp        [12] RevReqContent,           --Revocation Request
 *
 */
class SGCI_PKIMessage_Body_RevocationRequestMessage
    extends SGCI_PKIMessage_Body_RevocationRequestContent
{
    /**
     * Returns the XML representation of the CertReqMessages
     *
     * @param string $nodeName The name of the root node of the
         XML
     *
     * @throws SGCI_Exception if the CertReqMessages is empty
     * @return DOMElement
     */
    public function getXMLEncoded ($nodeName = 'rr')
    {
        return parent::getXMLEncoded($nodeName);
    }
}
```

Código Fonte C.30: Classe SGCI_PKIMessage_Body_RevocationRequestMessage

```
<?php
/**
 * Class that implements a RevRepContent defined in RFC4210.
 *
 * RevRepContent ::= SEQUENCE {
 * status        SEQUENCE SIZE (1..MAX) OF PKIStatusInfo,
 * -- in same order as was sent in RevReqContent
 * revCerts [0] SEQUENCE SIZE (1..MAX) OF CertId OPTIONAL,-- IDs
      for which revocation was requested
 * (same order as status)
 * crls      [1] SEQUENCE SIZE (1..MAX) OF CertificateList
     OPTIONAL
 * -- the resulting CRLs (there may be more than one)
 */
class SGCI_PKIMessage_Body_RevocationResponseContent extends
    SGCI_PKIMessage_Body
{
    /**
     * The status of the requested revocation certificates.
     *
     * @var array of integer
     * @see SGCI_PKIStatus
     */
    protected $_status = array();

    /**
     * The information about the certificates for witch
         revocation was requested.
     *
     * This array should look like:
     *
     * revCerts[]['issuer'] = GN;
     * revCerts[]['serialNumber'] = integer;
     *
     * @var array
     */
    protected $_revCerts = array();

    /**
     * The resulting CRLs.
     *
     * @var array of
         LabsecCL_Security_Certification_CertificateRevocationListCL

     */
    protected $_crls = array();

    const ISSUER = 'issuer';
    const SERIAL_NUMBER = 'serialNumber';

    /**
     * Loads an XML
```

```
     *
     * @param DOMNode $xml The XML to be loaded
     *
     * @return void
     */
    public function loadXML (DomNode $xml)
    {
        $status = $xml->getElementsByTagName('status')->item(0);
        $status = $status->getElementsByTagName('PKIStatusInfo')
            ;
        foreach ($status as $statusInfo) {
            $pkiStatus = new SGCI_PKIStatusInfo();
            $pkiStatus->loadXML($statusInfo);

            $this->_status[] = $pkiStatus;
            $this->_revCerts[] = null;
        }
        $revCerts = $xml->getElementsByTagName('revCerts')->item
            (0);
        if ($revCerts !== null) {
            $revCerts = $revCerts->getElementsByTagName('CertId'
                );
            foreach ($revCerts as $index => $revCert) {
                $revCertEntry = null;
                if (($revCert !== null) && ($revCert->nodeValue
                    !== '')) {
                    $revCertEntry = array();

                    $issuer = $revCert->getElementsByTagName(
                        self::ISSUER)->item(0);
                    $rdns = new
                        Labsec_Security_Certification_DataTypes_DirectoryName
                        ();
                    $rdns->loadXML($issuer);
                    $revCertEntry[self::ISSUER] = $rdns;

                    $serialNumber = $revCert->
                        getElementsByTagName(self::
                        SERIAL_NUMBER)->item(0);
                    $revCertEntry[self::SERIAL_NUMBER] =
                        $serialNumber->nodeValue;
                }
                $this->_revCerts[$index] = $revCertEntry;
            }
        }

        $crls = $xml->getElementsByTagName('crls')->item(0);
        if ($crls !== null) {
            $crls = $crls->getElementsByTagName('CertificateList
                ');
            foreach ($crls as $crl) {
                $crl = new
```

```
                    LabsecCL_Security_Certification_CertificateRevocation
                    (
                    $crl->nodeValue
                );
                $this->addCrl($crl);
            }
        }
    }

    /**
     * Adds a revocation response to the message
     *
     * @param SGCI_PKIStatusInfo
         $status       The status of the
     *
         response of this
     *
         specific certificate
     * @param LabsecCL_Security_Certification_CertificateCL
         $certificate The certificate for
     *
         which revocation was
     *
         requested
     *
     * @throws SGCI_Exception_InvalidParameter If the status is
         invalid
     * @return void
     */
    public function addRevocationResponse
    (
        SGCI_PKIStatusInfo $status,
        $certificate = null
    ) {
        if ($status->getStatus() === null) {
            throw new SGCI_Exception_InvalidParameter('status
                must not be empty');
        }
        $this->_status[] = $status;
        $revCert = null;
        if ($certificate !== null) {
            $revCert = $this->_getCertId($certificate);
        }
        $this->_revCerts[] = $revCert;
    }

    /**
     * Gets the issuer and serial number of a certificate object
```

```
     *
     * @param LabsecCL_Security_Certification_CertificateCL
          $certificate The certificate to get
     *
          the identification
     *
     * @return array in the following format:
     * array['issuer'] = GN;
     * array['serialNumber'] = integer;
     */
    protected function _getCertId(
        LabsecCL_Security_Certification_CertificateCL
        $certificate)
    {
        $revCert = array();
        $revCert[self::ISSUER] = $certificate->getIssuer();
        $revCert[self::SERIAL_NUMBER] = $certificate->
            getSerialNumber();

        return $revCert;
    }

    /**
     * Returns all the status
     *
     * @return array of SGCI_PKIStatusInfo
     */
    public function getStatus ()
    {
        return $this->_status;
    }

    /**
     * Returns all the RevCerts
     *
     * @return array
     */
    public function getRevCerts ()
    {
        return $this->_revCerts;
    }

    /**
     * Adds a CRL to the revocation response
     *
     * @param
          LabsecCL_Security_Certification_CertificateRevocationListCL
           $crl The crl to be added
     *
          to this response
```

```
     *
     * @return void
     */
    public function addCrl
    (
        LabsecCL_Security_Certification_CertificateRevocationListCL
            $crl
    ) {
        $this->_crls[] = $crl;
    }

    /**
     * Returns all the crls
     *
     * @return array of
         LabsecCL_Security_Certification_CertificateRevocationListCL

     */
    public function getCrls ()
    {
        return $this->_crls;
    }

    /**
     * Returns the XML representation of the CertRepMessage
     *
     * @param string $nodeName The name of the root node of the
         XML
     *
     * @throws SGCI_Exception If some of the required fields are
         empty
     * @return DOMElement
     */
    public function getXMLEncoded ($nodeName = 'RevRepContent')
    {
        if (empty($this->_status) === true) {
            throw new SGCI_Exception(
                'status is required and must have at least one
                    entry to generate XML'
            );
        }
        $xml = new DomDocument('1.0');

        $node = $xml->createElement($nodeName);
        $statusXML = $xml->createElement('status');
        foreach ($this->_status as $status) {
            $element = $status->getXMLEncoded();
            $element = $xml->importNode($element, true);
            $statusXML->appendChild($element);
        }

        $node->appendChild($statusXML);
```

```
        /*
            Verifies if exists an entry of a revCert that is not
                null,
            otherwise this field should not be present in XML
        */

        $comparator = array();
        $comparator[] = null;
        $array = array_diff($this->_revCerts, $comparator);
        if (empty($array) === false) {
            $revCertXML = $xml->createElement('revCerts');
            foreach ($this->_revCerts as $revCert) {
                $certId = $xml->createElement('CertId');
                if ($revCert !== null) {
                    $dn = new
                        Labsec_Security_Certification_DataTypes_DirectoryName
                        ();
                    $dn->addDirectoryName($revCert[self::ISSUER
                        ]);
                    $issuerXML = $dn->getXMLEncoded('issuer');
                    $issuerXML = $xml->importNode($issuerXML,
                        true);
                    $certId->appendChild($issuerXML);
                    $serialNumber = $xml->createElement(
                        self::SERIAL_NUMBER, $revCert[self::
                            SERIAL_NUMBER]
                    );
                    $certId->appendChild($serialNumber);
                }
                $revCertXML->appendChild($certId);
            }
            $node->appendChild($revCertXML);
        }
        if (empty($this->_crls) === false) {
            $crls = $xml->createElement('crls');
            foreach ($this->_crls as $crl) {
                $element = $xml->createElement('CertificateList'
                    , $crl->getPem());
                $crls->appendChild($element);
            }
            $node->appendChild($crls);
        }
        return $node;
    }
}
```

Código Fonte C.31: Classe SGCI_PKIMessage_Body_RevocationResponseContent

```
<?php
```

```
/**
 * Class that implements a rp defined as a choice of the PKIBody
      defined in RFC4210.
 *
 * rr        [11] RevRepContent,           --Revocation Response
 *
 */
class SGCI_PKIMessage_Body_RevocationResponseMessage
    extends SGCI_PKIMessage_Body_RevocationResponseContent
{
    /**
     * Returns the XML representation of the CertReqMessages
     *
     * @param string $nodeName The name of the root node of the
          XML
     *
     * @throws SGCI_Exception If some of the required fields are
          empty
     * @return DOMElement
     */
    public function getXMLEncoded ($nodeName = 'rp')
    {
        return parent::getXMLEncoded($nodeName);
    }
}
```

Código Fonte C.32: Classe SGCI_PKIMessage_Body_RevocationResponseMessage

```
<?php
/**
 * Class that implements a trusted relationship request. This
     class should be used as a new
 * choice option of the PKIBody structure, defined in RFC4210.
 *
 * TrustedRelReq ::= SEQUENCE {
 *      iPAddress       [0]     OCTET STRING,
 *  certificate [1] Certificate
 * }
 *
 */
class SGCI_PKIMessage_Body_TrustedRelationshipRequest extends
    SGCI_PKIMessage_Body
{
    /**
     * The IP Address of the requester.
     *
     * @var Labsec_Security_Certification_DataTypes_IPAddress
     */
    protected $_iPAddress;
```

```
    /**
     * The certificate of the requester.
     *
     * @var Labsec_Security_Certification_Certificate
     */
    protected $_certificate;

    /**
     * The certificate that will be used to sign the CMP
         messages.
     *
     * @var Labsec_Security_Certification_Certificate
     */
    protected $_transportCertificate;

   /**
    * Loads an XML
    *
    * @param DOMNode $xml The XML to be loaded
    *
    * @return void
    */
    public function loadXML (DOMNode $xml)
    {
        $iPAddressXml = $xml->getElementsByTagName('iPAddress')
            ->item(0);
        $iPAddress = new
            Labsec_Security_Certification_DataTypes_IPAddress()
            ;
        $iPAddress->setIPAddress($iPAddressXml->nodeValue);

        $certificate = $xml->getElementsByTagName('certificate')
            ->item(0);
        $certificate = $certificate->nodeValue;
        $certificate = new
            LabsecCL_Security_Certification_CertificateCL(
            $certificate);

        $this->setRequester($iPAddress, $certificate);

        $certificate = $xml->getElementsByTagName('
            transportCertificate')->item(0);
        if ($certificate !== null) {
            $certificate = $certificate->nodeValue;
            $certificate = new
                LabsecCL_Security_Certification_CertificateCL(
                $certificate);
            $this->setTransportCertificate($certificate);
        }
    }

    /**
```

```
     * Sets the requester of the trusted relationship
     *
     * @param Labsec_Security_Certification_DataTypes_IPAddress
         $iPAddress    The IP Address of the
     *
         requester
     * @param Labsec_Security_Certification_Certificate
         $certificate The certificate of
     *
         the requester
     *
     * @return void
     */
    public function setRequester
    (
        Labsec_Security_Certification_DataTypes_IPAddress
            $iPAddress,
        Labsec_Security_Certification_Certificate $certificate
    ) {
        $this->_iPAddress = $iPAddress;
        $this->_certificate = $certificate;
    }

    /**
     * Returns the ip address of the requester
     *
     * @return Labsec_Security_Certification_DataTypes_IPAddress
     */
    public function getIpAddress()
    {
        return $this->_iPAddress;
    }

    /**
     * Returns the certificate of the requester
     *
     * @return Labsec_Security_Certification_Certificate
     */
    public function getCertificate()
    {
        return $this->_certificate;
    }

    /**
     * Sets the certificate that will be used to sign messages
         of the CMP
     *
     * @param Labsec_Security_Certification_Certificate
         $certificate The certificate that will be
     *
```

```
     used to verify the signature
 *
     of the messages.
 *
 * @return void
 */
public function setTransportCertificate(
    Labsec_Security_Certification_Certificate $certificate)
{
    $this->_transportCertificate = $certificate;
}

/**
 * Gets the certificate that will be used to sign messages
     of the CMP
 *
 * @return Labsec_Security_Certification_Certificate
 */
public function getTransportCertificate()
{
    return $this->_transportCertificate;
}

/**
 * Returns the XML representation of the PKIBody
 *
 * @param string $nodeName The name of the root node of the
     XML
 *
 * @throws SGCI_Exception If the IpAddress or the
     certificate was not provided
 * @return DOMElement
 */
public function getXMLEncoded ($nodeName = 'trr')
{
    if (($this->_iPAddress === null) || ($this->_certificate
         === null)) {
        throw new SGCI_Exception(
            'The IP Address and the certificate must be
                provided to generate XML'
        );
    }

    $xml = new DOMDocument();

    $trustedRelReq = $xml->createElement($nodeName);

    $iPAddress = $xml->createElement('iPAddress', $this->
        _iPAddress->getIPAddress());
    $trustedRelReq->appendChild($iPAddress);
```

```
        $certificate = $xml->createElement('certificate', $this
            ->_certificate->getPem());
        $trustedRelReq->appendChild($certificate);

        $certificate = $this->_transportCertificate;
        if ($certificate !== null) {
            $certificate = $certificate->getPem();
            $certificate = $xml->createElement('
                transportCertificate', $certificate);
            $trustedRelReq->appendChild($certificate);
        }

        return $trustedRelReq;
    }
}
```

Código Fonte C.33: Classe SGCI_PKIMessage_Body_TrustedRelationshipRequest

```
<?php
/**
 * Class that implements a trusted relationship response.
 *
 * This class should be used as a new
 * choice option of the PKIBody structure, defined in RFC4210.
 *
 * TrustedRelRep ::= SEQUENCE {
 *      status       [0] PKIStatusInfo,
 *  certificate [1] Certificate OPTIONAL
 * }
 *
 */
class SGCI_PKIMessage_Body_TrustedRelationshipResponse extends
    SGCI_PKIMessage_Body
{
    /**
     * The status of the response.
     *
     * @var SGCI_PKIStatusInfo
     */
    protected $_status;

    /**
     * The certificate of the response.
     *
     * @var Labsec_Security_Certification_Certificate
     */
    protected $_certificate;

    /**
```

```
     * The certificate that will be used to sign the CMP
         messages.
     *
     * @var Labsec_Security_Certification_Certificate
     */
    protected $_transportCertificate;

    /**
     * Loads an XML
     *
     * @param DOMNode $xml The XML to be loaded
     *
     * @return void
     */
    public function loadXML (DOMNode $xml)
    {
        $status = $xml->getElementsByTagName('status')->item(0);
        $statusInfo = new SGCI_PKIStatusInfo();
        $statusInfo->loadXML($status);
        $this->setStatus($statusInfo);

        $certificate = $xml->getElementsByTagName('certificate')
            ->item(0);
        if ($certificate !== null) {
            $certificate = $certificate->nodeValue;
            $certificate = new
                LabsecCL_Security_Certification_CertificateCL(
                $certificate);
            $this->setCertificate($certificate);
        }

        $certificate = $xml->getElementsByTagName('
            transportCertificate')->item(0);
        if ($certificate !== null) {
            $certificate = $certificate->nodeValue;
            $certificate = new
                LabsecCL_Security_Certification_CertificateCL(
                $certificate);
            $this->setTransportCertificate($certificate);
        }
    }

    /**
     * Sets the status of the trusted relationship
     *
     * @param SGCI_PKIStatusInfo $status The status of the
         trusted relationship
     *
     * @throws SGCI_Exception_InvalidParameter If the status is
         empty
     * @return void
     */
```

```
public function setStatus (SGCI_PKIStatusInfo $status)
{
    if ($status->getStatus() !== null) {
        $this->_status = $status;
    } else {
        throw new SGCI_Exception_InvalidParameter('status
            must not be empty');
    }
}

/**
 * Returns the status of the trusted relationship
 *
 * @return SGCI_PKIStatusInfo
 */
public function getStatus ()
{
    return $this->_status;
}

/**
 * Sets the certificate of the response
 *
 * @param Labsec_Security_Certification_Certificate
     $certificate The certificate of the
 *

     response
 *
 * @throws SGCI_Exception If the status was not yet set or
     is not ACCEPTED status
 * @return void
 *
 */
public function setCertificate
(
    Labsec_Security_Certification_Certificate $certificate
) {
    if ($this->_status === null) {
        throw new SGCI_Exception(
            'Certificate cannot be setted. Status is not
                setted.'
        );

    }
    $status = $this->_status->getStatus();
    if ($status === SGCI_PKIStatus::STATUS_ACCEPTED) {
        $this->_certificate = $certificate;
    } else {
        throw new SGCI_Exception(
            'Certificate cannot be setted. status should be
                ACCEPTED'
```

```
            );
        }
    }

    /**
     * Returns the certificate of the response
     *
     * @return Labsec_Security_Certification_Certificate
     */
    public function getCertificate ()
    {
        return $this->_certificate;
    }

        /**
     * Sets the certificate that will be used to sign messages
         of the CMP
     *
     * @param Labsec_Security_Certification_Certificate
         $certificate The certificate that will be
     *
         used to verify the signature
     *
         of the messages.
     *
     * @return void
     */
    public function setTransportCertificate(
        Labsec_Security_Certification_Certificate $certificate)
    {
        $this->_transportCertificate = $certificate;
    }

    /**
     * Gets the certificate that will be used to sign messages
         of the CMP
     *
     * @return Labsec_Security_Certification_Certificate
     */
    public function getTransportCertificate()
    {
        return $this->_transportCertificate;
    }

    /**
     * Returns the XML representation of the PKIBody
     *
     * @param string $nodeName The name of the root node of the
         XML
     *
```

```
     * @throws SGCI_Exception If the status was not provided
     * @return DOMElement
     */
    public function getXMLEncoded ($nodeName = 'trp')
    {
        if ($this->_status === null) {
            throw new SGCI_Exception(
                'status is required and must be provided to
                    generate XML'
            );
        }

        $xml = new DOMDocument();

        $trustedRelRep = $xml->createElement($nodeName);

        $status = $this->_status->getXMLEncoded('status');
        $status = $xml->importNode($status, true);
        $trustedRelRep->appendChild($status);

        if ($this->_certificate !== null) {
            $certificate = $xml->createElement('certificate',
                $this->_certificate->getPem());
            $trustedRelRep->appendChild($certificate);
        }

         $certificate = $this->_transportCertificate;
         if ($certificate !== null) {
             $certificate = $certificate->getPem();
             $certificate = $xml->createElement('
                 transportCertificate', $certificate);
             $trustedRelReq->appendChild($certificate);
         }

        return $trustedRelRep;
    }
}
```

Código Fonte C.34: Classe SGCI_PKIMessage_Body_TrustedRelationshipResponse

```
<?php
/**
 * Class that tests the SGCI_PKIFailureInfo class
 */
class SGCI_PKIFailureInfoTest extends PHPUnit_Framework_TestCase
{
    /**
     * Test for CanVerifyStatus
     *
     * @param integer $failureInfo The failure provided by the
         failureInfoProvider
```

```
     *
     * @dataProvider failureInfoProvider
     * @return void
     */
    public function testCanVerifyStatus ($failureInfo)
    {
        $valid = SGCI_PKIFailureInfo::verifyFailure($failureInfo
            );
        $this->assertTrue($valid);
    }

    /**
     * DataPrivder with all correct failures
     *
     * @return void
     */
    public function failureInfoProvider ()
    {
        //TODO utilizar reflection
        return array(
                array(SGCI_PKIFailureInfo::
                    FAILURE_ADD_INFO_NOT_AVALIABLE),
                array(SGCI_PKIFailureInfo::FAILURE_BAD_ALGOTIRHM
                    ),
                array(SGCI_PKIFailureInfo::FAILURE_BAD_CERT_ID),
                array(SGCI_PKIFailureInfo::
                    FAILURE_BAD_CERT_TEMPLATE),
                array(SGCI_PKIFailureInfo::
                    FAILURE_BAD_DATA_FORMAT),
                array(SGCI_PKIFailureInfo::
                    FAILURE_BAD_MESSAGE_CHECK),
                array(SGCI_PKIFailureInfo::FAILURE_BAD_POP),
                array(SGCI_PKIFailureInfo::
                    FAILURE_BAD_RECIPIENT_NONCE),
                array(SGCI_PKIFailureInfo::FAILURE_BAD_REQUEST),
                array(SGCI_PKIFailureInfo::
                    FAILURE_BAD_SENDER_NONCE),
                array(SGCI_PKIFailureInfo::FAILURE_BAD_TIME),
                array(SGCI_PKIFailureInfo::
                    FAILURE_CERT_CONFIRMED),
                array(SGCI_PKIFailureInfo::FAILURE_CERT_REVOKED)
                    ,
                array(SGCI_PKIFailureInfo::
                    FAILURE_DUPLICATE_CERT_REQ),
                array(SGCI_PKIFailureInfo::
                    FAILURE_INCORRECT_DATA),
                array(SGCI_PKIFailureInfo::
                    FAILURE_MISSING_TIMESTAMP),
                array(SGCI_PKIFailureInfo::
                    FAILURE_NOT_AUTHORIZED),
                array(SGCI_PKIFailureInfo::
                    FAILURE_SIGNER_NOT_TRUSTED),
```

```
                array(SGCI_PKIFailureInfo::
                    FAILURE_SYSTEM_FAILURE),
                array(SGCI_PKIFailureInfo::
                    FAILURE_SYSTEM_UNAVAIL),
                array(SGCI_PKIFailureInfo::
                    FAILURE_TIME_NOT_AVALIABLE),
                array(SGCI_PKIFailureInfo::
                    FAILURE_TRANSACTION_ID_IN_USE),
                array(SGCI_PKIFailureInfo::
                    FAILURE_UNACCEPTED_EXTENSION),
                array(SGCI_PKIFailureInfo::
                    FAILURE_UNACCEPTED_POLICY),
                array(SGCI_PKIFailureInfo::
                    FAILURE_UNSUPPORTED_VERSION),
                array(SGCI_PKIFailureInfo::
                    FAILURE_WRONG_AUTHORITY),
                array(SGCI_PKIFailureInfo::
                    FAILURE_WRONG_INTEGRITY),
            );
    }

    /**
     * Test for CanVerifyStatus
     *
     * @param mixed $failureInfo Wrong failures provided by the
         wrongFailureInfoProvider
     *
     * @dataProvider wrongFailureInfoProvider
     * @return void
     */
    public function testVerifyStatusWithInvalidStatus (
        $failureInfo)
    {
        $valid = SGCI_PKIFailureInfo::verifyFailure($failureInfo
            );
        $this->assertFalse($valid);
    }

    /**
     * DataProvider with incorrect failures
     *
     * @return void
     */
    public function wrongFailureInfoProvider ()
    {
        return array(
                array(30),
                array('7'),
                array('invalid'),
                array(array('a')),
            );
    }
```

```
}
```

Código Fonte C.35: Classe SGCI_PKIFailureInfoTest

```
<?php
/**
 * Class that tests the SGCI_PKIMessage class
 */
class SGCI_PKIMessageTest extends PHPUnit_Framework_TestCase
{
    /**
     * Class being tested.
     *
     * @var SGCI_PKIMessage
     */
    protected $_pkiMessage;

    /**
     * Instantiate a SGCI_PKIMessage object
     *
     * @return void
     */
    public function setUp ()
    {
        $this->_pkiMessage = new SGCI_PKIMessage();
    }

    /**
     * Test for loadXML
     *
     * @return void
     */
    public function
        testLoadXMLWithInvalidXMLShouldThrowAnException ()
    {
        $this->setExpectedException('SGCI_Exception');

        $xml = '<PKIMessage><header><teste>teste</teste></header
            ><body></body></PKIMessage>';
        $this->_pkiMessage->loadXML($xml);
    }

    /**
     * Test for loadXML
     *
     * @return void
     */
    public function testLoadXMLWithOnlyRequiredFields ()
    {
        $expectedHeader = self::getHeader();
        $expectedBody = self::getBody();
```

```
        $xml = self::getXML($expectedHeader, $expectedBody);
        $this->_pkiMessage->loadXML($xml);

        $this->assertEquals($expectedHeader, $this->_pkiMessage
            ->getHeader());
        $this->assertEquals($expectedBody, $this->_pkiMessage->
            getBody());
    }

    /**
     * Test for loadXML
     *
     * @return void
     */
    public function testLoadXMLWithAllFields()
    {
        $keyPair = self::getKeyPair();
        $hashAlg = LabsecCL_Security_Crypto_MessageDigestCL::
            ALGORITHM_SHA512;

        $protectionAlg =
        Labsec_Security_Certification_ObjectIdentifierFactory::
            getObjectIdentifierByAlgorithm(
            $keyPair->getPrivateKey()->getAlgorithm(), $hashAlg
        );

        $expectedHeader = self::getHeader();
        $expectedBody = self::getBody();

        $this->_pkiMessage->setBody($expectedBody);
        $this->_pkiMessage->setHeader($expectedHeader);

        $signer = new SGCI_PKIMessageSigner();
        $this->_pkiMessage = $signer->sign($this->_pkiMessage,
            $hashAlg, $keyPair->getPrivateKey());

        $expectedSignature = $this->_pkiMessage->getSignature();
        $expectedHeader->setProtectionAlg($protectionAlg);

        $xml = self::getXML($expectedHeader, $expectedBody,
            $expectedSignature);

        $this->_pkiMessage = new SGCI_PKIMessage();
        $this->_pkiMessage->loadXML($xml);

        $this->assertEquals($expectedHeader, $this->_pkiMessage
            ->getHeader());
        $this->assertEquals($expectedBody, $this->_pkiMessage->
            getBody());
        $this->assertEquals($expectedSignature, $this->
            _pkiMessage->getSignature());
    }
```

```
    /**
     * Test for setHeader
     *
     * @return void
     */
    public function testCanSetHeader ()
    {
        $header = new SGCI_PKIMessage_Header();
        $this->_pkiMessage->setHeader($header);
        $this->assertEquals($header, $this->_pkiMessage->
            getHeader());
    }

    /**
     * Test for setBody
     *
     * @return void
     */
    public function testCanSetBody ()
    {
        $body = new
            SGCI_PKIMessage_Body_CertificationRequestMessage();
        $this->_pkiMessage->setBody($body);
        $this->assertEquals($body, $this->_pkiMessage->getBody()
            );
    }

    /**
     * Test for setSignature
     *
     * @return void
     */
    public function testCanSetSignature ()
    {
        $signature = '12345';

        $this->_pkiMessage->setSignature($signature);

        $this->assertEquals($signature, $this->_pkiMessage->
            getSignature());
    }

    /**
     * Test for setSignature
     *
     * @param mixed $signature A wrong value for a signature
         provided by the dataProvider
     *
     * @dataProvider wrongSignatureProvider
     * @return void
     */
```

```
public function
    testSetSignatureWithinvalidValueShouldThrowException (
    $signature)
{
    $this->setExpectedException('
        SGCI_Exception_InvalidParameter');

    $this->_pkiMessage->setSignature($signature);
}

/**
 * DataProvider with incorrect status
 *
 * @return void
 */
public function wrongSignatureProvider ()
{
    return array(
             array(10),
             array(new DOMDocument()),
             array(array(1)),
            );
}
/**
 * Constructs a key pair
 *
 * @param string $keyAlgorithm The algorithm of the key
 *
 * @return LabsecCL_Security_Crypto_KeyPairCL
 */
public static function getKeyPair ($keyAlgorithm = 'ECDSA')
{
    $builder = new LabsecCL_Security_Crypto_KeyPairBuilderCL
        ($keyAlgorithm);
    return $builder->getKeyPair();
}

/**
 * Test for getXMLEncoded
 *
 * @return void
 */
public function testGetXMLEncodedWithoutHeader ()
{
    $this->setExpectedException('SGCI_Exception');
    $body = new
        SGCI_PKIMessage_Body_CertificationRequestMessage();
    $this->_pkiMessage->setBody($body);
    $this->_pkiMessage->getXMLEncoded();
}

/**
```

```
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithoutBody ()
    {
        $this->setExpectedException('SGCI_Exception');
        $header = new SGCI_PKIMessage_Header();
        $this->_pkiMessage->setHeader($header);
        $this->_pkiMessage->getXMLEncoded();
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncoded ()
    {
        $header = self::getHeader();
        $body = self::getBody();
        $this->_pkiMessage->setHeader($header);
        $this->_pkiMessage->setBody($body);

        $expectedXML = self::getXML($header, $body);
        $actualXMl = $this->_pkiMessage->getXMLEncoded();

        $dom = new DOMDocument();
        $dom->loadXML($actualXMl);
        $this->assertEquals($expectedXML, $actualXMl);
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithSignedMessage ()
    {
        $header = self::getHeader();
        $body = self::getBody();
        $this->_pkiMessage->setHeader($header);
        $this->_pkiMessage->setBody($body);

        $keyPair = self::getKeyPair();
        $hashAlg = LabsecCL_Security_Crypto_MessageDigestCL::
            ALGORITHM_SHA256;

        $expectedProtectionAlg =
        Labsec_Security_Certification_ObjectIdentifierFactory::
            getObjectIdentifierByAlgorithm(
            $keyPair->getPrivateKey()->getAlgorithm(), $hashAlg
```

```
        );

        $header->setProtectionAlg($expectedProtectionAlg);

        $signer = new SGCI_PKIMessageSigner();
        $signer->sign($this->_pkiMessage, $hashAlg, $keyPair->
            getPrivateKey());

        $signature = $this->_pkiMessage->getSignature();

        $expectedXML = self::getXML($header, $body, $signature);
        $actualXML = $this->_pkiMessage->getXMLEncoded();

        $dom = new DOMDocument();
        $dom->loadXML($actualXML);
        $this->assertEquals($expectedXML, $actualXML);
    }

    /**
     * Returns the body of a PKIMessage
     *
     * @return SGCI_PKIMessage_Body
     */
    public static function getBody ()
    {
        $certTemplate = new SGCI_PKIMessage_CertificateTemplate
            ();
        $certTemplate->setSerialNumber(2);
        $body = new
            SGCI_PKIMessage_Body_CertificationRequestMessage();
        $certReq = new
            SGCI_PKIMessage_CertificationRequestMessage();
        $certReq->setCertReqId(5);
        $certReq->setCertTemplate($certTemplate);
        $body->addCertReqMsg($certReq);

        return $body;
    }

    /**
     * Returns the header of a PKIMessage
     *
     * @return SGCI_PKIMessage_Header
     */
    public static function getHeader ()
    {
        $header = new SGCI_PKIMessage_Header();
        $gn = new
            Labsec_Security_Certification_DataTypes_DirectoryName
            ();
        $sender = new
            Labsec_Security_Certification_DataTypes_RDNSequence
```

```
            ();
        $sender->addEntry('CN', 'Sender');
        $gn->addDirectoryName($sender);
        $header->setSender($gn);
        $gn = new
            Labsec_Security_Certification_DataTypes_DirectoryName
            ();
        $recipient = new
            Labsec_Security_Certification_DataTypes_RDNSequence
            ();
        $recipient->addEntry('CN', 'Recipient');
        $gn->addDirectoryName($recipient);
        $header->setRecipient($gn);

        return $header;
    }

    /**
     * Construct the XML from the parameters and return a string
          representation of the XML
     *
     * @param SGCI_PKIMessage_Header $header     The header of
        the PKIMessage
     * @param SGCI_PKIMessage_Body   $body       The body of the
        PKIMessage
     * @param string                 $signature The signature of
          the PKIMessage
     *
     * @return string
     */
    public static function getXML
    (
        SGCI_PKIMessage_Header $header,
        SGCI_PKIMessage_Body $body,
        $signature = null
    ) {
        $xml = new DOMDocument();
        $pkiMessage = $xml->createElement('PKIMessage');

        $headerXML = $header->getXMLEncoded('header');
        $headerXML = $xml->importNode($headerXML, true);
        $pkiMessage->appendChild($headerXML);

        $bodyElement = $xml->createElement('body');
        $bodyXML = $body->getXMLEncoded();
        $bodyXML = $xml->importNode($bodyXML, true);
        $bodyElement->appendChild($bodyXML);
        $pkiMessage->appendChild($bodyElement);

        if ($signature !== null) {
            $signature = base64_encode($signature);
            $protection = $xml->createElement('protection',
```

```
                $signature);
            $pkiMessage->appendChild($protection);
        }

        $xml->appendChild($pkiMessage);

        return $xml->saveXML();
    }
}
```

Código Fonte C.36: Classe SGCI_PKIMessageTest

```
<?php
/**
 * Class that tests the SGCI_PKIMessageSigner class
 *
 */
class SGCI_PKIMessageSignerTest extends
    PHPUnit_Framework_TestCase
{
    /**
     * Class being tested.
     *
     * @var SGCI_PKIMessageSigner
     */
    protected $_signer;

    /**
     * Instantiate a SGCI_PKIMessageSigner object
     *
     * @return void
     */
    public function setUp ()
    {
        $this->_signer = new SGCI_PKIMessageSigner();
    }

    /**
     * Test for sign
     *
     * @return void
     */
    public function testSignWithoutBodyShoulThrowAnException ()
    {
        $this->setExpectedException('SGCI_Exception');

        $body = new
            SGCI_PKIMessage_Body_CertificationRequestMessage();
        $pkiMessage = new SGCI_PKIMessage();
        $pkiMessage->setBody($body);

        $this->_signer->sign(
```

```
            $pkiMessage, 'SHA1', SGCI_PKIMessageTest::getKeyPair
                ()->getPrivateKey()
        );
    }

    /**
     * Test for sign
     *
     * @return void
     */
    public function testSignWithoutHeaderShoulThrowAnException
        ()
    {
        $this->setExpectedException('SGCI_Exception');

        $header = new SGCI_PKIMessage_Header();
        $pkiMessage = new SGCI_PKIMessage();
        $pkiMessage->setHeader($header);

        $this->_signer->sign(
            $pkiMessage, 'SHA1', SGCI_PKIMessageTest::getKeyPair
                ()->getPrivateKey()
        );
    }

    /**
     * Test for sign
     *
     * @param string $hashAlgorithm The hash to be used for
         signing the PKIMessage
     * @param string $keyAlgorithm  The key algorithm to be used
          for signing the PKIMessage
     *
     * @dataProvider SignAlgorithmProvider
     * @return void
     */
    public function testCanSign ($hashAlgorithm, $keyAlgorithm)
    {
        $header = SGCI_PKIMessageTest::getHeader();
        $body = SGCI_PKIMessageTest::getBody();
        $pkiMessage = new SGCI_PKIMessage();

        $pkiMessage->setHeader($header);
        $pkiMessage->setBody($body);

        $keyPair = SGCI_PKIMessageTest::getKeyPair($keyAlgorithm
            );

        $expectedProtectionAlg =
        Labsec_Security_Certification_ObjectIdentifierFactory::
            getObjectIdentifierByAlgorithm(
            $keyPair->getPrivateKey()->getAlgorithm(),
```

```
                $hashAlgorithm
        );

        $header->setProtectionAlg($expectedProtectionAlg);

        $pkiMessage = $this->_signer->sign($pkiMessage,
            $hashAlgorithm, $keyPair->getPrivateKey());

        $this->assertTrue($this->_signer->verify($pkiMessage,
            $keyPair->getPublicKey()));

        $actualProtectionAlg = $pkiMessage->getHeader()->
            getProtectionAlg();
        $this->assertEquals($expectedProtectionAlg,
            $actualProtectionAlg);
    }

    /**
     * Data provider for SignAlgorithmProvider
     *
     * @return array
     */
    public function signAlgorithmProvider()
    {
        return array(
                array(
                 LabsecCL_Security_Crypto_MessageDigestCL::
                     ALGORITHM_SHA1,
                 Labsec_Security_Crypto_AsymmetricKey::
                     ALGORITHM_ECDSA,
                ),
                array(
                 LabsecCL_Security_Crypto_MessageDigestCL::
                     ALGORITHM_SHA224,
                 Labsec_Security_Crypto_AsymmetricKey::
                     ALGORITHM_ECDSA,
                ),
                array(
                 LabsecCL_Security_Crypto_MessageDigestCL::
                     ALGORITHM_SHA256,
                 Labsec_Security_Crypto_AsymmetricKey::
                     ALGORITHM_ECDSA,
                ),
                array(
                 LabsecCL_Security_Crypto_MessageDigestCL::
                     ALGORITHM_SHA384,
                 Labsec_Security_Crypto_AsymmetricKey::
                     ALGORITHM_ECDSA,
                ),
                array(
                 LabsecCL_Security_Crypto_MessageDigestCL::
                     ALGORITHM_SHA512,
```

```
                Labsec_Security_Crypto_AsymmetricKey::
                    ALGORITHM_ECDSA,
                ),
                array(
                 LabsecCL_Security_Crypto_MessageDigestCL::
                    ALGORITHM_SHA1,
                 Labsec_Security_Crypto_AsymmetricKey::
                    ALGORITHM_RSA,
                ),
                array(
                 LabsecCL_Security_Crypto_MessageDigestCL::
                    ALGORITHM_SHA224,
                 Labsec_Security_Crypto_AsymmetricKey::
                    ALGORITHM_RSA,
                ),
                array(
                 LabsecCL_Security_Crypto_MessageDigestCL::
                    ALGORITHM_SHA256,
                 Labsec_Security_Crypto_AsymmetricKey::
                    ALGORITHM_RSA,
                ),
                array(
                 LabsecCL_Security_Crypto_MessageDigestCL::
                    ALGORITHM_SHA384,
                 Labsec_Security_Crypto_AsymmetricKey::
                    ALGORITHM_RSA,
                ),
                array(
                 LabsecCL_Security_Crypto_MessageDigestCL::
                    ALGORITHM_SHA512,
                 Labsec_Security_Crypto_AsymmetricKey::
                    ALGORITHM_RSA,
                ),
               );
    }

    /**
     * Test for verify
     *
     * @return void
     */
    public function
        testVerifySignatureWithoutBodyShoulThrowAnException ()
    {
        $this->setExpectedException('SGCI_Exception');

        $body = new
            SGCI_PKIMessage_Body_CertificationRequestMessage();
        $pkiMessage = new SGCI_PKIMessage();
        $pkiMessage->setBody($body);

        $this->_signer->verify($pkiMessage, SGCI_PKIMessageTest
```

```
            ::getKeyPair()->getPublicKey());
    }

    /**
     * Test for verify
     *
     * @return void
     */
    public function
        testVerifySignatureWithoutHeaderShoulThrowAnException
        ()
    {
        $this->setExpectedException('SGCI_Exception');

        $header = new SGCI_PKIMessage_Header();
        $pkiMessage = new SGCI_PKIMessage();
        $pkiMessage->setHeader($header);

        $this->_signer->verify(
            $pkiMessage, SGCI_PKIMessageTest::getKeyPair()->
                getPublicKey()
        );
    }
}
```

Código Fonte C.37: Classe SGCI_PKIMessageSignerTest

```
<?php
/**
 * Class that tests the SGCI_PKIStatus class
 */
class SGCI_PKIStatusTest extends PHPUnit_Framework_TestCase
{
    /**
     * Test for VerifyStatus
     *
     * @param integer $status A status provided by the
         statusProvider
     *
     * @dataProvider statusProvider
     * @return void
     */
    public function testCanVerifyStatus ($status)
    {
        $valid = SGCI_PKIStatus::verifyStatus($status);
        $this->assertTrue($valid);
    }

    /**
     * DataProvider with correct status
     *
     * @return void
```

```
     */
    public function statusProvider ()
    {
        return array(
                array(SGCI_PKIStatus::STATUS_ACCEPTED),
                array(SGCI_PKIStatus::STATUS_GRANTED_WITH_MODS),
                array(SGCI_PKIStatus::STATUS_KEY_UPDATE_WARNING)
                    ,
                array(SGCI_PKIStatus::STATUS_REJECTION),
                array(SGCI_PKIStatus::
                    STATUS_REVOCATION_NOTIFICATION),
                array(SGCI_PKIStatus::STATUS_REVOCATION_WARNING)
                    ,
                array(SGCI_PKIStatus::STATUS_WAITING),
               );
    }

    /**
     * Test for VerifyStatus
     *
     * @param mixed $status A status provided by
         wrongStatusProvider
     *
     * @dataProvider wrongStatusProvider
     * @return void
     */
    public function testVerifyStatusWithInvalidStatus ($status)
    {
        $valid = SGCI_PKIStatus::verifyStatus($status);
        $this->assertFalse($valid);
    }

    /**
     * DataProvider with incorrect status
     *
     * @return void
     */
    public function wrongStatusProvider ()
    {
        return array(
                array(10),
                array('7'),
                array(array(1)),
               );
    }
}
```

Código Fonte C.38: Classe SGCI_PKIStatusTest

```
<?php
/**
 * Class that tests the SGCI_PKIStatusInfo class
```

```
 */
class SGCI_PKIStatusInfoTest extends PHPUnit_Framework_TestCase
{
    /**
     * Class being tested.
     *
     * @var SGCI_PKIStatusInfo
     */
    protected $_statusInfo;

    /**
     * Instantiate a SGCI_PKIStatusInfo object
     *
     * @return void
     */
    public function setUp ()
    {
        $this->_statusInfo = new SGCI_PKIStatusInfo();
    }

    /**
     * Test for loadXML
     *
     * @return void
     */
    public function testLoadXMLWithOnlyRequiredFields ()
    {
        $status = SGCI_PKIStatus::STATUS_ACCEPTED;
        $xml = $this->_getDomNode($status);
        $this->_statusInfo->loadXML($xml);
        $this->assertEquals($status, $this->_statusInfo->
            getStatus());
        $this->assertNull($this->_statusInfo->getStatusString())
            ;
        $this->assertNull($this->_statusInfo->getFailInfo());
    }

    /**
     * Test for loadXML
     *
     * @return void
     */
    public function testLoadXMLAllFields ()
    {
        $status = SGCI_PKIStatus::STATUS_REJECTION;
        $statusString = 'this is a status string for the
            rejection status';
        $failInfo = SGCI_PKIFailureInfo::
            FAILURE_BAD_CERT_TEMPLATE;
        $xml = $this->_getDomNode($status, $statusString,
            $failInfo);
        $this->_statusInfo->loadXML($xml);
```

```
        $this->assertEquals($status, $this->_statusInfo->
            getStatus());
        $this->assertEquals($statusString, $this->_statusInfo->
            getStatusString());
        $this->assertEquals($failInfo, $this->_statusInfo->
            getFailInfo());
    }

    /**
     * Construct the XML from the parameters and return a
         DomNode representation of the XML
     *
     * @param integer $status       The status
     * @param string  $statusString A human readable information
          for status
     * @param integer $failInfo     A failure information for
         status
     *
     * @return DOMNode
     */
    protected function _getDomNode ($status, $statusString =
        null, $failInfo = null)
    {
        $xml = $this->_getXML($status, $statusString, $failInfo)
            ;
        $dom = new DOMDocument();
        $dom->loadXML($xml);
        return $dom->getElementsByTagName('PKIStatusInfo')->item
            (0);
    }

    /**
     * Test for setStatus
     *
     * @return void
     */
    public function testCanSetStatus ()
    {
        $expectedStatus = SGCI_PKIStatus::STATUS_ACCEPTED;
        $this->_statusInfo->setStatus($expectedStatus);
        $actualStatus = $this->_statusInfo->getStatus();
        $this->assertEquals($expectedStatus, $actualStatus);
    }

    /**
     * Test for setStatus
     *
     * @return void
     */
    public function testSetInvalidStatusShouldThrowException ()
    {
        $this->setExpectedException('
```

```
            SGCI_Exception_InvalidParameter');
        $this->_statusInfo->setStatus('InvalidStatus');
    }

    /**
     * Test for setStatusString
     *
     * @return void
     */
    public function testSetStatusString ()
    {
        $expectedStatusString = 'This is the expected status
            string';
        $this->_statusInfo->setStatusString(
            $expectedStatusString);
        $actualStatusString = $this->_statusInfo->
            getStatusString();
        $this->assertEquals($expectedStatusString,
            $actualStatusString);
    }

    /**
     * Test for setStatusString
     *
     * @return void
     */
    public function
        testSetInvalidStatusStringShouldThrowException ()
    {
        $this->setExpectedException('
            SGCI_Exception_InvalidParameter');
        $this->_statusInfo->setStatusString(array());
    }

    /**
     * Test for setFailInfo
     *
     * @return void
     */
    public function testCanSetFailInfo ()
    {
        $this->_statusInfo->setStatus(SGCI_PKIStatus::
            STATUS_REJECTION);
        $expectedFailInfo = SGCI_PKIFailureInfo::
            FAILURE_BAD_CERT_ID;
        $this->_statusInfo->setFailInfo($expectedFailInfo);
        $actualFailInfo = $this->_statusInfo->getFailInfo();
        $this->assertEquals($expectedFailInfo, $actualFailInfo);
    }

    /**
     * Test for setFailInfo
```

```
     *
     * @return void
     */
    public function testSetInvalidFailInfoShouldThrowException
        ()
    {
        $this->setExpectedException('
            SGCI_Exception_InvalidParameter');
        $this->_statusInfo->setStatus(SGCI_PKIStatus::
            STATUS_REJECTION);
        $this->_statusInfo->setFailInfo('InvalidFailInfo');
    }

    /**
     * Test for setFailInfo
     *
     * @return void
     */
    public function
        testSetFailInfoWithNotRejectionStatusShouldThrowException
         ()
    {
        $this->setExpectedException('SGCI_Exception');
        $this->_statusInfo->setStatus(SGCI_PKIStatus::
            STATUS_ACCEPTED);
        $expectedFailInfo = SGCI_PKIFailureInfo::
            FAILURE_BAD_CERT_ID;
        $this->_statusInfo->setFailInfo($expectedFailInfo);
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncondedWithOnlyRequiredFields ()
    {
        $status = SGCI_PKIStatus::STATUS_ACCEPTED;
        $expectedXML = $this->_getXML($status);
        $this->_statusInfo->setStatus($status);
        $pkiStatusInfo = $this->_statusInfo->getXMLEncoded();
        $xml = new DomDocument();
        $pkiStatusInfoXML = $xml->importNode($pkiStatusInfo,
            true);
        $xml->appendChild($pkiStatusInfoXML);
        $this->assertTrue($xml->schemaValidateSource($this->
            _getPKIStatusInfoXMLSchema()));
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Test for getXMLEncoded
```

```
     *
     * @return void
     */
    public function testGetXMLEncondedWithAllFields ()
    {
        $status = SGCI_PKIStatus::STATUS_REJECTION;
        $statusString = 'This is a status string';
        $failInfo = SGCI_PKIFailureInfo::FAILURE_CERT_REVOKED;
        $expectedXML = $this->_getXML($status, $statusString,
            $failInfo);
        $this->_statusInfo->setStatus($status);
        $this->_statusInfo->setStatusString($statusString);
        $this->_statusInfo->setFailInfo($failInfo);
        $pkiStatusInfo = $this->_statusInfo->getXMLEncoded();
        $xml = new DomDocument();
        $pkiStatusInfoXML = $xml->importNode($pkiStatusInfo,
            true);
        $xml->appendChild($pkiStatusInfoXML);
        $this->assertTrue($xml->schemaValidateSource($this->
            _getPKIStatusInfoXMLSchema()));
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function
        testGetXMLEncodedWithoutStatusShouldThrowException ()
    {
        $this->setExpectedException('SGCI_Exception');
        $this->_statusInfo->getXMLEncoded();
    }

    /**
     * Construct the XML from the parameters and return a string
          representation of the XML
     *
     * @param integer $status       The status
     * @param string  $statusString A human readable information
          for status
     * @param integer $failInfo     A failure information for
         status
     *
     * @return string
     */
    protected function _getXML ($status, $statusString = null,
        $failInfo = null)
    {
        $statusStringXML = '';
        $failInfoXML = '';
```

```
        if ($statusString !== null) {
            $statusStringXML = '<statusString>' . $statusString
                . '</statusString>';
        }
        if ($failInfo !== null) {
            $failInfoXML = '<failInfo>' . $failInfo . '</
                failInfo>';
        }
        $xml = '<?xml version="1.0"?>' . PHP_EOL . '<
            PKIStatusInfo>' .
         '<status>' . $status . '</status>' . $statusStringXML .
              $failInfoXML .
         '</PKIStatusInfo>' . PHP_EOL;
        return $xml;
    }

    /**
     * Returns the XML schema for the PKIStatusInfo structure
     *
     * @return string
     */
    protected function _getPKIStatusInfoXMLSchema ()
    {
        return '<?xml version="1.0"?>
                        <xs:schema xmlns:xs="http://www.w3.org
                            /2001/XMLSchema">

                        <xs:element name="PKIStatusInfo">
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element name
                                                    ="status"
                                                    type="xs:
                                                    integer"/>
                                                <xs:element name
                                                    ="
                                                    statusString
                                                    " type="xs:
                                                    string"
                                                    minOccurs
                                                    ="0"/>
                                                <xs:element name
                                                    ="failInfo"
                                                     type="xs:
                                                    integer"
                                                    minOccurs
                                                    ="0"/>
                                        </xs:sequence>
                                </xs:complexType>
                        </xs:element>

                        </xs:schema>';
```

```
    }
}
```

Código Fonte C.39: Classe SGCI_PKIStatusInfoTest

```
<?php

require_once __DIR__ . '/Body/InitializationRequestMessageTest.
    php';

/**
 * Class that tests the SGCI_PKIMessage_BodyFactory class
 */
class SGCI_PKIMessage_BodyFactoryTest extends
    PHPUnit_Framework_TestCase
{
    /**
     * Test for getPKIBodyByXML
     *
     * @return void
     */
    public function testCanGetInitializationRequestByXML ()
    {
        $expectedBody =
         SGCI_PKIMessage_Body_InitializationRequestMessageTest::
             getInitializationRequestMessage();

        $body = SGCI_PKIMessage_BodyFactory::getPKIBodyByXML(
            $expectedBody->getXMLEncoded());

        $this->assertEquals($expectedBody, $body);
    }

    /**
     * Test for getPKIBodyByXML
     *
     * @return void
     */
    public function testCanGetInitializationResponseByXML ()
    {
        $expectedBody =
         SGCI_PKIMessage_Body_InitializationResponseMessageTest
             ::getInitializationResponseMessage(
         );

        $body = SGCI_PKIMessage_BodyFactory::getPKIBodyByXML(
            $expectedBody->getXMLEncoded());

        $this->assertEquals($expectedBody, $body);
    }

    /**
```

```
     * Test for getPKIBodyByXML
     *
     * @return void
     */
    public function testCanGetCertificationRequestByXML ()
    {
        $expectedBody =
            SGCI_PKIMessage_Body_CertificationRequestMessageTest
                ::getCertificationRequestMessage();

        $body = SGCI_PKIMessage_BodyFactory::getPKIBodyByXML(
            $expectedBody->getXMLEncoded());

        $this->assertEquals($expectedBody, $body);
    }

    /**
     * Test for getPKIBodyByXML
     *
     * @return void
     */
    public function testCanGetCertificationResponseByXML ()
    {
        $expectedBody =
         SGCI_PKIMessage_Body_CertificationResponseMessageTest::
              getCertificationResponseMessage();

        $body = SGCI_PKIMessage_BodyFactory::getPKIBodyByXML(
            $expectedBody->getXMLEncoded());

        $this->assertEquals($expectedBody, $body);
    }

    /**
     * Test for getPKIBodyByXML
     *
     * @return void
     */
    public function testCanGetRevocationRequestByXML ()
    {
        $expectedBody =
            SGCI_PKIMessage_Body_RevocationRequestMessageTest::
                getRevocationRequestMessage();

        $body = SGCI_PKIMessage_BodyFactory::getPKIBodyByXML(
            $expectedBody->getXMLEncoded());

        $this->assertEquals($expectedBody, $body);
    }

    /**
     * Test for getPKIBodyByXML
```

```
     *
     * @return void
     */
    public function testCanGetRevocationResponseByXML ()
    {
        $expectedBody =
            SGCI_PKIMessage_Body_RevocationResponseMessageTest::
                getRevocationResponseMessage();

        $body = SGCI_PKIMessage_BodyFactory::getPKIBodyByXML(
            $expectedBody->getXMLEncoded());

        $this->assertEquals($expectedBody, $body);
    }

    /**
     * Test for getPKIBodyByXML
     *
     * @return void
     */
    public function testCanGetListCertsReqByXML ()
    {
        $expectedBody = new
            SGCI_PKIMessage_Body_ListCertificatesRequest();

        $body = SGCI_PKIMessage_BodyFactory::getPKIBodyByXML(
            $expectedBody->getXMLEncoded());

        $this->assertEquals($expectedBody, $body);
    }

    /**
     * Test for getPKIBodyByXML
     *
     * @return void
     */
    public function testCanGetListCertsRepByXML ()
    {
        $expectedBody = new
            SGCI_PKIMessage_Body_ListCertificatesResponse();

        $body = SGCI_PKIMessage_BodyFactory::getPKIBodyByXML(
            $expectedBody->getXMLEncoded());

        $this->assertEquals($expectedBody, $body);
    }

    /**
     * Test for getPKIBodyByXML
     *
     * @return void
     */
```

```
    public function testCanGetTrustedRelReqByXML ()
    {
        $expectedBody = $this->_getTrustedRelReq();

        $body = SGCI_PKIMessage_BodyFactory::getPKIBodyByXML(
            $expectedBody->getXMLEncoded());

        $this->assertEquals($expectedBody, $body);
    }

    /**
     * Returns a SGCI_PKIMessage_Body_TrustedRelationshipRequest
         object
     *
     * @return SGCI_PKIMessage_Body_TrustedRelationshipRequest
     */
    protected function _getTrustedRelReq ()
    {
        $iPAddress = new
            Labsec_Security_Certification_DataTypes_IPAddress()
            ;
        $iPAddress->setIPAddress('192.168.66.5');

        $trustedRelReq = new
            SGCI_PKIMessage_Body_TrustedRelationshipRequest();
        $trustedRelReq->setRequester(
            $iPAddress,
                SGCI_PKIMessage_Body_TrustedRelationshipRequestTest
                ::getCertificate()
        );

        return $trustedRelReq;
    }

    /**
     * Test for getPKIBodyByXML
     *
     * @return void
     */
    public function testCanGetTrustedRelRepByXML ()
    {
        $expectedBody = $this->_getTrustedRelRep();

        $body = SGCI_PKIMessage_BodyFactory::getPKIBodyByXML(
            $expectedBody->getXMLEncoded());

        $this->assertEquals($expectedBody, $body);
    }

    /**
     * Returns a
         SGCI_PKIMessage_Body_TrustedRelationshipResponse
```

```
            object
     *
     * @return SGCI_PKIMessage_Body_TrustedRelationshipResponse
     */
    protected function _getTrustedRelRep ()
    {
        $status = new SGCI_PKIStatusInfo();
        $status->setStatus(SGCI_PKIStatus::STATUS_REJECTION);

        $trustedRelRep = new
            SGCI_PKIMessage_Body_TrustedRelationshipResponse();
        $trustedRelRep->setStatus($status);

        return $trustedRelRep;
    }

    /**
     * Test for getPKIBodyByXML
     *
     * @return void
     */
    public function
        testGetPKIBodyByXmlWithInvalidBodyShouldThrowException
        ()
    {
        $this->setExpectedException('SGCI_Exception');

        $xml = '<invalid><test>test</test></invalid>';

        $domDoc = new DOMDocument();
        $domDoc->loadXML($xml);
        $node = $domDoc->getElementsByTagName('invalid')->item
            (0);

        SGCI_PKIMessage_BodyFactory::getPKIBodyByXML($node);
    }
}
```

Código Fonte C.40: Classe SGCI_PKIMessage_BodyFactoryTest

```
<?php
/**
 * Class that tests the SGCI_PKIMessage_CertificateTemplate
     class
 */
class SGCI_PKIMessage_CertificateTemplateTest extends
    PHPUnit_Framework_TestCase
{
    /**
     * Class being tested.
     *
     * @var SGCI_PKIMessage_CertificateTemplate
```

```
     */
    protected $_certTemplate;

    /**
     * Instantiate a SGCI_PKIMessage_CertificateTemplate object
     *
     * @return void
     */
    public function setUp ()
    {
        $this->_certTemplate = new
            SGCI_PKIMessage_CertificateTemplate();
    }

    /**
     * Test for loadXML
     *
     * @return void
     */
    public function testLoadXMLWithOnlyVersion ()
    {
        $version = 1;
        $xml = $this->_getDomNode($version);

        $this->_certTemplate->loadXML($xml);

        $this->assertEquals($version, $this->_certTemplate->
            getVersion());
        $this->assertNull($this->_certTemplate->getPublicKey());
        $this->assertNull($this->_certTemplate->getSerialNumber
            ());
        $this->assertNull($this->_certTemplate->getSubject());
    }

    /**
     * Test for loadXML
     *
     * @return void
     */
    public function testLoadXMLWithOnlySerialNumber ()
    {
        $serialNumber = 5;
        $xml = $this->_getDomNode(null, $serialNumber);

        $this->_certTemplate->loadXML($xml);

        $this->assertEquals($serialNumber, $this->_certTemplate
            ->getSerialNumber());
        $this->assertNull($this->_certTemplate->getPublicKey());
        $this->assertNull($this->_certTemplate->getVersion());
        $this->assertNull($this->_certTemplate->getSubject());
    }
```

```
/**
 * Test for loadXML
 *
 * @return void
 */
public function testLoadXMLWithOnlySubject ()
{
    $subject = new
        Labsec_Security_Certification_DataTypes_RDNSequence
        ();
    $subject->addEntry(
        Labsec_Security_Certification_DataTypes_RDNSequence
            ::COMMON_NAME,
        'CommonName'
    );
    $subject->addEntry(
        Labsec_Security_Certification_DataTypes_RDNSequence
            ::ORGANIZATION_UNIT,
        'OrganizationUnit'
    );

    $xml = $this->_getDomNode(null, null, $subject);

    $this->_certTemplate->loadXML($xml);

    $actualSubject = $this->_certTemplate->getSubject()->
        getEntries();
    $this->assertEquals($subject->getEntries(),
        $actualSubject);
    $this->assertNull($this->_certTemplate->getPublicKey());
    $this->assertNull($this->_certTemplate->getSerialNumber
        ());
    $this->assertNull($this->_certTemplate->getVersion());
}

/**
 * Test for loadXML
 *
 * @return void
 */
public function testLoadXMLWithOnlyPubKey ()
{
    $pubKey = $this->_getPublicKey()->getPemEncoded();

    $xml = $this->_getDomNode(null, null, null, $pubKey);

    $this->_certTemplate->loadXML($xml);

    $this->assertEquals($pubKey, $this->_certTemplate->
        getPublicKey()->getPemEncoded());
    $this->assertNull($this->_certTemplate->getVersion());
```

```
        $this->assertNull($this->_certTemplate->getSerialNumber
            ());
        $this->assertNull($this->_certTemplate->getSubject());
    }

    /**
     * Test for loadXML
     *
     * @return void
     */
    public function testLoadXMLWithAllFields()
    {
        $subject = new
            Labsec_Security_Certification_DataTypes_RDNSequence
            ();
        $subject->addEntry(
            Labsec_Security_Certification_DataTypes_RDNSequence
                ::COMMON_NAME,
            'CommonName'
        );
        $subject->addEntry(
            Labsec_Security_Certification_DataTypes_RDNSequence
                ::ORGANIZATION_UNIT,
            'OrganizationUnit'
        );

        $pubKey = $this->_getPublicKey()->getPemEncoded();

        $version = 0;
        $serialNumber = 25;
        $xml = $this->_getDomNode($version, $serialNumber,
            $subject, $pubKey);

        $this->_certTemplate->loadXML($xml);

        $this->assertEquals($version, $this->_certTemplate->
            getVersion());
        $this->assertEquals($serialNumber, $this->_certTemplate
            ->getSerialNumber());

        $actualSubject = $this->_certTemplate->getSubject()->
            getEntries();
        $this->assertEquals($subject->getEntries(),
            $actualSubject);

        $this->assertEquals($pubKey, $this->_certTemplate->
            getPublicKey()->getPemEncoded());
    }

    /**
     * Construct the XML from the parameters and return a
         DomNode representation of the XML
```

```
     *
     * @param integer
                                                        $version
              The version of the
     *
         certificate
     * @param integer
         $serialNumber The serial number
     *
         of the certificate
     * @param
         Labsec_Security_Certification_DataTypes_RDNSequence
         $subject       The subject of
     *
         the certificate
     * @param string
                                                        $pubKey
               PEM encoded of
     *
         the public key
     *
     * @return DOMNode
     */
    protected function _getDomNode
    (
        $version = null,
        $serialNumber = null,
        $subject = null,
        $pubKey = null
    ) {
        $xml = $this->_getXML($version, $serialNumber, $subject,
             $pubKey);

        $dom = new DOMDocument();
        $dom->loadXML($xml);

        return $dom->getElementsByTagName('CertTemplate')->item
            (0);
    }

    /**
     * Test for setVersion
     *
     * @return void
     */
    public function testCanSetVersion ()
    {
```

```
        $version = 1;
        $this->_certTemplate->setVersion($version);

        $this->assertEquals($version, $this->_certTemplate->
            getVersion());
    }

    /**
     * Test for setVersion
     *
     * @return void
     */
    public function testSetVersionWithNonInteger ()
    {
        $this->setExpectedException('
            SGCI_Exception_InvalidParameter');
        $this->_certTemplate->setVersion(array(21));

        $this->assertEquals($version, $this->_certTemplate->
            getVersion());
    }

    /**
     * Test for setVersion
     *
     * @return void
     */
    public function testSetVersionWithInvalidVersion ()
    {
        $this->setExpectedException('
            SGCI_Exception_InvalidParameter');

        $this->_certTemplate->setVersion(5);
    }

    /**
     * Test for setSerialNumber
     *
     * @return void
     */
    public function testCanSetSerialNumber ()
    {
        $serial = 1;
        $this->_certTemplate->setSerialNumber($serial);

        $this->assertEquals($serial, $this->_certTemplate->
            getSerialNumber());
    }

    /**
     * Test for setSerialNumber
     *
```

```
     * @return void
     */
    public function testSetSerialNumberWithInvalidData ()
    {
        $this->setExpectedException('
            SGCI_Exception_InvalidParameter');

        $this->_certTemplate->setSerialNumber('5');
    }

    /**
     * Test for setSubject
     *
     * @return void
     */
    public function testCanSetSubject ()
    {
        $subject = new
            Labsec_Security_Certification_DataTypes_RDNSequence
            ();
        $subject->addEntry('CN', 'Test');
        $this->_certTemplate->setSubject($subject);

        $this->assertEquals($subject, $this->_certTemplate->
            getSubject());
    }

    /**
     * Test for setSubject
     *
     * @return void
     */
    public function testSetSubjectWithEmptyDN ()
    {
        $this->setExpectedException('
            SGCI_Exception_InvalidParameter');

        $subject = new
            Labsec_Security_Certification_DataTypes_RDNSequence
            ();
        $this->_certTemplate->setSubject($subject);
    }

    /**
     * Test for setPublicKey
     *
     * @return void
     */
    public function testCanSetPublicKey ()
    {
        $pubKey = $this->_getPublicKey();
        $this->_certTemplate->setPublicKey($pubKey);
```

```
        $this->assertTrue(
            $this->_certTemplate->getPublicKey() instanceof
                Labsec_Security_Crypto_PublicKey
        );
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithOnlyVersionSetted ()
    {
        $version = 0;
        $expectedXML = $this->_getXML($version);

        $this->_certTemplate->setVersion($version);

        $certTemplateXML = $this->_certTemplate->getXMLEncoded()
            ;

        $xml = new DomDocument();

        $certTemplateXML = $xml->importNode($certTemplateXML,
            true);
        $xml->appendChild($certTemplateXML);

        $validSchema = $xml->schemaValidateSource($this->
            _getCertTemplateXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function
        testGetXMLWithEmptyCertTemplateShouldThrowException ()
    {
        $this->setExpectedException('SGCI_Exception');

        $this->_certTemplate->getXMLEncoded();
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
```

```
     */
    public function testGetXMLEncodedWithOnlySerialNumberSetted
        ()
    {
        $serialNumber = 2;
        $expectedXML = $this->_getXML(null, $serialNumber);

        $this->_certTemplate->setSerialNumber($serialNumber);

        $certTemplateXML = $this->_certTemplate->getXMLEncoded()
            ;

        $xml = new DomDocument();

        $certTemplateXML = $xml->importNode($certTemplateXML,
            true);

        $xml->appendChild($certTemplateXML);
        $validSchema = $xml->schemaValidateSource($this->
            _getCertTemplateXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithOnlySubjectSetted ()
    {
        $commonName = 'My Test';
        $subject = new
            Labsec_Security_Certification_DataTypes_RDNSequence
            ();
        $subject->addEntry('CN', $commonName);

        $expectedXML = $this->_getXML(null, null, $subject);

        $this->_certTemplate->setSubject($subject);

        $certTemplateXML = $this->_certTemplate->getXMLEncoded()
            ;

        $xml = new DomDocument();

        $certTemplateXML = $xml->importNode($certTemplateXML,
            true);
        $xml->appendChild($certTemplateXML);

        $validSchema = $xml->schemaValidateSource($this->
```

```
            _getCertTemplateXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithOnlyPublicKeySetted()
    {
        $publicKey = $this->_getPublicKey();
        $publicKeyPemEncoded = $publicKey->getPemEncoded();

        $expectedXML = $this->_getXML(null, null, null,
            $publicKeyPemEncoded);

        $this->_certTemplate->setPublicKey($publicKey);

        $certTemplateXML = $this->_certTemplate->getXMLEncoded()
            ;

        $xml = new DomDocument();
        $certTemplateXML = $xml->importNode($certTemplateXML,
            true);
        $xml->appendChild($certTemplateXML);

        $validSchema = $xml->schemaValidateSource($this->
            _getCertTemplateXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithAllValuesSetted()
    {
        $commonName = 'Test';
        $subject = new
            Labsec_Security_Certification_DataTypes_RDNSequence
            ();
        $subject->addEntry('CN', $commonName);

        $publicKey = $this->_getPublicKey();
        $publicKeyPemEncoded = $publicKey->getPemEncoded();
```

```
        $version = 1;
        $serialNumber = 5;
        $expectedXML = $this->_getXML($version, $serialNumber,
            $subject, $publicKeyPemEncoded);

        $this->_certTemplate->setVersion($version);
        $this->_certTemplate->setSerialNumber($serialNumber);
        $this->_certTemplate->setSubject($subject);
        $this->_certTemplate->setPublicKey($publicKey);

        $certTemplateXML = $this->_certTemplate->getXMLEncoded()
            ;

        $xml = new DomDocument();

        $certTemplateXML = $xml->importNode($certTemplateXML,
            true);
        $xml->appendChild($certTemplateXML);

        $validSchema = $xml->schemaValidateSource($this->
            _getCertTemplateXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Generates a public key
     *
     * @return Labsec_Security_Crypto_PublicKey
     */
    protected function _getPublicKey()
    {
        $keyBuilder = new
            LabsecCL_Security_Crypto_KeyPairBuilderCL(
            Labsec_Security_Crypto_AsymmetricKey::ALGORITHM_RSA
        );
        $keyBuilder->setSize(1024);
        $keyPair = $keyBuilder->getKeyPair();
        return $keyPair->getPublicKey();
    }

    /**
     * Construct the XML from the parameters and return a string
          representation of the XML
     *
     * @param integer
                                                    $version
              The version of the
     *

         certificate
```

```
     * @param integer
     *      $serialNumber The serial number
     *
     *      of the certificate
     * @param
     *      Labsec_Security_Certification_DataTypes_RDNSequence
     *      $subject      The subject of
     *
     *      the certificate
     * @param string
     *                                                      $pubKey
     *          PEM encoded of
     *
     *      the public key
     *
     * @return string
     */
    protected function _getXML
    (
        $version = null,
        $serialNumber = null,
        $subject = null,
        $pubKey = null
    ) {
        $versionXML = '';
        $serialNumberXML = '';
        $subjectXML = '';
        $pubKeyXML = '';
        if ($version !== null) {
            $versionXML = '<version>' . $version . '</version>';
        }
        if ($serialNumber !== null) {
            $serialNumberXML = '<serialNumber>' . $serialNumber
                . '</serialNumber>';
        }
        if ($subject !== null) {
            $subjectXML = $subject->getEntriesAsDomNode('subject
                ');
            $doc = new DOMDocument();
            $subjectXML = $doc->importNode($subjectXML, true);
            $subjectXML = $doc->saveXML($subjectXML);
        }
        if ($pubKey !== null) {
            $pubKeyXML = '<publicKey>' . $pubKey . '</publicKey>
                ';
        }
        $xml = '<?xml version="1.0"?>' . PHP_EOL . '<
            CertTemplate>' . $versionXML .
```

```
        $serialNumberXML . $subjectXML . $pubKeyXML . '</
            CertTemplate>' .
        PHP_EOL;
        return $xml;
    }
    /**
     * Returns the XML schema for the CertTemplate structure
     *
     * @return string
     */
    protected function _getCertTemplateXMLSchema ()
    {
        return '<?xml version="1.0"?>
                        <xs:schema xmlns:xs="http://www.w3.org
                            /2001/XMLSchema">
                        <xs:element name="CertTemplate">
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element ref
                                                    ="version"
                                                    minOccurs
                                                    ="0"/> <!--
                                                     Optional
                                                    -->
                                                <xs:element name
                                                    ="
                                                    serialNumber
                                                    " minOccurs
                                                    ="0" type="
                                                    xs:integer
                                                    "/> <!--
                                                    Optional
                                                    -->
                                                <xs:element ref
                                                    ="subject"
                                                    minOccurs
                                                    ="0"/> <!--
                                                     Optional
                                                    -->
                                                <xs:element name
                                                    ="publicKey
                                                    " minOccurs
                                                    ="0" type="
                                                    xs:string
                                                    "/> <!--
                                                    Optional
                                                    -->
                                        </xs:sequence>
                                </xs:complexType>
                        </xs:element>

                        <xs:element name="version"
```

```
    substitutionGroup="Version"/>

<xs:element name="Version">
        <xs:simpleType>
                <xs:restriction base="xs
                    :integer">
                        <xs:enumeration
                            value="0"/>
                             <!-- v1
                            -->
                        <xs:enumeration
                            value="1"/>
                             <!-- v2
                            -->
                        <xs:enumeration
                            value="2"/>
                             <!-- v3
                            -->
            </xs:restriction>
        </xs:simpleType>
</xs:element>

<xs:element name="subject"
    substitutionGroup="RDNSequence"/>

<xs:element name="RDNSequence">
        <xs:complexType>
                <xs:sequence>
                <xs:choice>
                        <xs:element name
                            ="C" type="
                            xs:string
                            "/> <!--
                            Country -->
                        <xs:element name
                            ="L" type="
                            xs:string
                            "/> <!--
                            Locality
                            -->
                        <xs:element name
                            ="ST" type
                            ="xs:string
                            "/> <!--
                            State or
                            province
                            -->
                        <xs:element name
                            ="O" type="
                            xs:string
                            "/> <!--
                            Organization
```

```
                                                        -->
                                                    <xs:element name
                                                        ="OU" type
                                                        ="xs:string
                                                        "/> <!--
                                                        Organization
                                                         Unit -->
                                                    <xs:element name
                                                        ="CN" type
                                                        ="xs:string
                                                        "/> <!--
                                                        Common Name
                                                         -->
                                                    <xs:element name
                                                        ="STREET"
                                                        type="xs:
                                                        string"/>
                                                        <!-- Street
                                                         Address
                                                        -->
                                                    <xs:element name
                                                        ="E" type="
                                                        xs:string
                                                        "/> <!-- E-
                                                        mail
                                                        Address -->
                                            </xs:choice>
                                            </xs:sequence>
                                    </xs:complexType>
                            </xs:element>
                    </xs:schema>';
    }
}
```

Código Fonte C.41: Classe SGCI_PKIMessage_CertificateTemplateTest

```
<?php
/**
 * Class that tests the
     SGCI_PKIMessage_CertificationRequestMessage class
 */
class SGCI_PKIMessage_CertificationRequestMessageTest extends
    PHPUnit_Framework_TestCase
{
    /**
     * Class being tested.
     *
     * @var SGCI_PKIMessage_Body_CertificationRequestMessage
     */
    protected $_certRequest;

    /**
```

```
     * Instantiate a SGCI_PKIMessage_CertificationRequestMessage
         object.
     *
     * @return void
     */
    public function setUp ()
    {
        $this->_certRequest = new
            SGCI_PKIMessage_CertificationRequestMessage();
    }
    /**
     * Test for loadXML
     *
     * @return void
     */
    public function testLoadXML ()
    {
        $certTemplate = new SGCI_PKIMessage_CertificateTemplate
            ();
        $certTemplate->setSerialNumber(1);

        $certReqId = 1;
        $xml = $this->_getDomNode($certReqId, $certTemplate);

        $this->_certRequest->setCertReqId($certReqId);
        $this->_certRequest->setCertTemplate($certTemplate);

        $this->_certRequest->loadXML($xml);

        $this->assertEquals($certReqId, $this->_certRequest->
            getCertReqId());
        $this->assertEquals($certTemplate, $this->_certRequest->
            getCertTemplate());
    }

    /**
     * Construct the XML from the parameters and return a
         DomNode representation of the XML
     *
     * @param integer                                $certReqId
         The id of the request
     * @param SGCI_PKIMessage_CertificateTemplate $certTemplate
         The template for the request
     *
     * @return void
     */
    protected function _getDomNode
    (
        $certReqId,
        SGCI_PKIMessage_CertificateTemplate $certTemplate
    ) {
        $xml = $this->_getXML($certReqId, $certTemplate);
```

```
        $dom = new DOMDocument();
        $dom->loadXML($xml);

        return $dom->getElementsByTagName('CertReqMsg')->item(0)
            ;
    }

    /**
     * Test for setCertReqId
     *
     * @return void
     */
    public function testCanSetCertReqId ()
    {
        $id = 1;
        $this->_certRequest->setCertReqId($id);

        $this->assertEquals($id, $this->_certRequest->
            getCertReqId());
    }

    /**
     * Test for setCertReqId
     *
     * @return void
     */
    public function testSetCertReqIdWithInvalidId ()
    {
        $this->setExpectedException('
            SGCI_Exception_InvalidParameter');

        $this->_certRequest->setCertReqId(array(5));
    }

    /**
     * Test for setCertTemplate
     *
     * @return void
     */
    public function testCanSetCertTemplate ()
    {
        $this->_certRequest->setCertTemplate(new
            SGCI_PKIMessage_CertificateTemplate());

        $this->assertTrue(
            $this->_certRequest->getCertTemplate() instanceof
                SGCI_PKIMessage_CertificateTemplate
        );
    }

    /**
```

```
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function
        testGetXMLEncodedWithoutCertReqIdShouldThrowException
        ()
    {
        $this->setExpectedException('SGCI_Exception');

        $this->_certRequest->setCertTemplate(new
            SGCI_PKIMessage_CertificateTemplate());
        $this->_certRequest->getXMLEncoded();
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function
        testGetXMLEncodedWithoutCertTemplateShouldThrowException
         ()
    {
        $this->setExpectedException('SGCI_Exception');

        $this->_certRequest->setCertReqId(1);
        $this->_certRequest->getXMLEncoded();
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncoded ()
    {
        $certTemplate = new SGCI_PKIMessage_CertificateTemplate
            ();
        $certTemplate->setSerialNumber(5);

        $certReqId = 1;
        $expectedXML = $this->_getXML($certReqId, $certTemplate)
            ;

        $this->_certRequest->setCertReqId($certReqId);
        $this->_certRequest->setCertTemplate($certTemplate);

        $certReqXML = $this->_certRequest->getXMLEncoded();

        $xml = new DomDocument();
```

```
        $certReqXML = $xml->importNode($certReqXML, true);
        $xml->appendChild($certReqXML);

        $validSchema = $xml->schemaValidateSource($this->
            _getCertReqMsgXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Construct the XML from the parameters and return a string
         representation of the XML
     *
     * @param integer                                   $certReqId
         The id of the request
     * @param SGCI_PKIMessage_CertificateTemplate $certTemplate
         The template for the request
     *
     * @return void
     */
    protected function _getXML
    (
        $certReqId,
        SGCI_PKIMessage_CertificateTemplate $certTemplate
    ) {
        $certTemplateXML = $certTemplate->getXMLEncoded('
            certTemplate');

        $doc = new DOMDocument();

        $certTemplateXML = $doc->importNode($certTemplateXML,
            true);
        $certTemplateXML = $doc->saveXML($certTemplateXML);

        $xml = '<?xml version="1.0"?>' . PHP_EOL
            . '<CertReqMsg>' . '<certReq>'
            . '<certReqId>' . $certReqId . '</certReqId>' .
                $certTemplateXML . '</certReq>'
            . '</CertReqMsg>' . PHP_EOL;

        return $xml;
    }

    /**
     * Returns the XML schema for the CertRequest structure
     *
     * @return string
     */
    protected function _getCertReqMsgXMLSchema ()
    {
        return '<?xml version="1.0"?>
```

```
<xs:schema xmlns:xs="http://www.w3.org
    /2001/XMLSchema">
<xs:element name="CertReqMsg">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref
                            ="certReq
                            "/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="certReq"
    substitutionGroup="CertRequest"/>

<xs:element name="CertRequest">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name
                            ="certReqId
                            " type="xs:
                            integer"/>
                        <xs:element ref
                            ="
                            certTemplate
                            "/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="certTemplate"
    substitutionGroup="CertTemplate"/>

<xs:element name="CertTemplate">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref
                            ="version"
                            minOccurs
                            ="0"/> <!--
                             Optional
                            -->
                        <xs:element name
                            ="
                            serialNumber
                            " minOccurs
                            ="0" type="
                            xs:integer
                            "/> <!--
                            Optional
                            -->
                        <xs:element ref
```

```
                                    ="subject"
                                    minOccurs
                                    ="0"/> <!--
                                     Optional
                                    -->
                                <xs:element name
                                    ="publicKey
                                    " minOccurs
                                    ="0" type="
                                    xs:string
                                    "/> <!--
                                    Optional
                                    -->
                        </xs:sequence>
                </xs:complexType>
        </xs:element>

        <xs:element name="version"
            substitutionGroup="Version"/>

        <xs:element name="Version">
                <xs:simpleType>
                        <xs:restriction base="xs
                            :integer">
                                <xs:enumeration
                                    value="0"/>
                                     <!-- v1
                                    -->
                                <xs:enumeration
                                    value="1"/>
                                     <!-- v2
                                    -->
                                <xs:enumeration
                                    value="2"/>
                                     <!-- v3
                                    -->
                    </xs:restriction>
                </xs:simpleType>
        </xs:element>

        <xs:element name="subject"
            substitutionGroup="RDNSequence"/>

        <xs:element name="RDNSequence">
                <xs:complexType>
                        <xs:sequence>
                                <xs:choice>
                                        <xs:
                                            element

                                            name
                                            ="C
```

```
                                                    "
                                                    type
                                                    ="
                                                    xs:
                                                    string
                                                    "/>

                                                    <!--

                                                    Country

                                                    -->
                                                <xs:
                                                    element

                                                    name
                                                    ="L
                                                    "
                                                    type
                                                    ="
                                                    xs:
                                                    string
                                                    "/>

                                                    <!--

                                                    Locality

                                                    -->
                                                <xs:
                                                    element

                                                    name
                                                    ="
                                                    ST"

                                                    type
                                                    ="
                                                    xs:
                                                    string
                                                    "/>

                                                    <!--

                                                    State
                                                     or

                                                    province

                                                    -->
                                                <xs:
                                                    element
```

```
                name
                ="O
                "
                type
                ="
                xs:
                string
                "/>

                <!--

                Organization

                -->
            <xs:
                element

                name
                ="
                OU"

                type
                ="
                xs:
                string
                "/>

                <!--

                Organization

                Unit

                -->
            <xs:
                element

                name
                ="
                CN"

                type
                ="
                xs:
                string
                "/>

                <!--

                Common
```

```
                                                    Name

                                                    -->
                                                <xs:
                                                    element

                                                    name
                                                    ="
                                                    STREET
                                                    "
                                                    type
                                                    ="
                                                    xs:
                                                    string
                                                    "/>

                                                    <!--

                                                    Street

                                                    Address

                                                    -->
                                                <xs:
                                                    element

                                                    name
                                                    ="E
                                                    "
                                                    type
                                                    ="
                                                    xs:
                                                    string
                                                    "/>

                                                    <!--
                                                     E-
                                                    mail

                                                    Address

                                                    -->
                                            </xs:choice>
                                        </xs:sequence>
                                    </xs:complexType>
                                </xs:element>
                                </xs:schema>';
    }
}
```

Código Fonte C.42: Classe

SGCI_PKIMessage_CertificationRequestMessageTest

```
<?php
/**
 * Class that tests the SGCI_PKIMessage_Header class
 */
class SGCI_PKIMessage_HeaderTest extends
    PHPUnit_Framework_TestCase
{
    /**
     * Class being tested.
     *
     * @var SGCI_PKIMessage_Header
     */
    protected $_header;

    /**
     * The sender of the message.
     *
     * @var
         Labsec_Security_Certification_DataTypes_DirectoryName
     */
    protected $_sender;

    /**
     * The recipient of the message.
     *
     * @var
         Labsec_Security_Certification_DataTypes_DirectoryName
     */
    protected $_recipient;

    /**
     * Instantiate a SGCI_PKIMessage_CertificationRequestMessage
          object and
     * sets the sender and recipient
     *
     * @return void
     */
    public function setUp ()
    {
        $this->_header = new SGCI_PKIMessage_Header();

        $this->_sender = new
            Labsec_Security_Certification_DataTypes_DirectoryName
            ();
        $rdns = new
            Labsec_Security_Certification_DataTypes_RDNSequence
            ();
        $rdns->addEntry(
            Labsec_Security_Certification_DataTypes_RDNSequence
                ::COMMON_NAME,
```

```
            'senderCN'
        );
        $this->_sender->addDirectoryName($rdns);

        $this->_recipient = new
            Labsec_Security_Certification_DataTypes_DirectoryName
            ();
        $rdns = new
            Labsec_Security_Certification_DataTypes_RDNSequence
            ();
        $rdns->addEntry(
            Labsec_Security_Certification_DataTypes_RDNSequence
                ::COMMON_NAME,
            'recipientCN'
        );
        $this->_recipient->addDirectoryName($rdns);
    }

    /**
     * Test for loadXML
     *
     * @return void
     */
    public function testLoadXMLWithOnlyRequiredFields ()
    {
        $xml = $this->_getDomNode($this->_sender, $this->
            _recipient);

        $this->_header->loadXML($xml);

        $this->assertEquals(
            $this->_sender->getDirectoryNames(),
            $this->_header->getSender()->getDirectoryNames()
        );

        $this->assertEquals(
            $this->_recipient->getDirectoryNames(),
            $this->_header->getRecipient()->getDirectoryNames()
        );

        $this->assertNull($this->_header->getMessageTime());
        $this->assertNull($this->_header->getRecipientKeyId());
        $this->assertNull($this->_header->getRecipientNonce());
        $this->assertNull($this->_header->getSenderKeyId());
        $this->assertNull($this->_header->getSenderNonce());
        $this->assertNull($this->_header->getTransactionId());
        $this->assertNull($this->_header->getProtectionAlg());
    }

    /**
     * Test for loadXML
     *
```

```
     * @return void
     */
    public function testLoadXMLWithAllFields ()
    {
        $recipNonce = 'Recp1234';
        $senderNonce = 'Sender1234';
        $transactionID = '4';
        $recipKID = 'RKID1234';
        $senderKID = 'SKID1234';
        $time = new Zend_Date();
        $messageTime = $time->getTimestamp();

        $protectionAlg =
            Labsec_Security_Certification_ObjectIdentifierFactory
                ::getObjectIdentifierByAlgorithm(
                Labsec_Security_Crypto_AsymmetricKey::
                    ALGORITHM_RSA,
                LabsecCL_Security_Crypto_MessageDigestCL::
                    ALGORITHM_SHA1
            );

        $xml = $this->_getDomNode(
            $this->_sender, $this->_recipient, $messageTime,
                $protectionAlg,
            $senderKID, $recipKID, $transactionID, $senderNonce,
                 $recipNonce
        );

        $this->_header->loadXML($xml);

        $this->assertEquals(
            $this->_sender->getDirectoryNames(),
            $this->_header->getSender()->getDirectoryNames()
        );

        $this->assertEquals(
            $this->_recipient->getDirectoryNames(),
            $this->_header->getRecipient()->getDirectoryNames()
        );

        $this->assertEquals($messageTime, $this->_header->
            getMessageTime()->getTimestamp());
        $this->assertEquals($recipKID, $this->_header->
            getRecipientKeyId());
        $this->assertEquals($recipNonce, $this->_header->
            getRecipientNonce());
        $this->assertEquals($senderKID, $this->_header->
            getSenderKeyId());
        $this->assertEquals($senderNonce, $this->_header->
            getSenderNonce());
        $this->assertEquals($transactionID, $this->_header->
            getTransactionId());
```

```
        $this->assertEquals($protectionAlg->getOID(),
            $protectionAlg->getOID());
    }

    /**
     * Construct the XML from the parameters and return a
       DomNode representation of the XML
     *
     * @param
       Labsec_Security_Certification_DataTypes_DirectoryName
       $sender          The sender of
     *
                                                          the
       message
     * @param
       Labsec_Security_Certification_DataTypes_DirectoryName
       $recipient       The recipient
     *
                                                          of the
        message
     * @param string
       $messageTime     The time of
     *
       production of
     *
                                                          the
       message
     * @param string
       $protectionAlg   The protection
     *
                              algorithm used to
     *
                              sign the message
     * @param string
       $senderKID       The sender
     *
                                                          key
       identifier
     * @param string
       $recipientKID    The recipient
```

```
     *
                                                                key
         identifier
     * @param string

         $transactionId  The transaction
     *

         id
     * @param string

         $senderNonce    The nonce of
     *

         the sender
     * @param string

         $recipientNonce The nonce of
     *

         the recipient
     *
     * @return DOMNode
     */
    protected function _getDomNode
    (
        Labsec_Security_Certification_DataTypes_DirectoryName
            $sender,
        Labsec_Security_Certification_DataTypes_DirectoryName
            $recipient,
        $messageTime = null,
        $protectionAlg = null,
        $senderKID = null,
        $recipientKID = null,
        $transactionId = null,
        $senderNonce = null,
        $recipientNonce = null
    ) {
        $xml = $this->_getXML(
            $sender, $recipient, $messageTime, $protectionAlg,
                $senderKID,
            $recipientKID, $transactionId, $senderNonce,
                $recipientNonce
        );
        $dom = new DOMDocument();
        $dom->loadXML($xml);
        return $dom->getElementsByTagName('PKIHeader')->item(0);
    }

    /**
     * Test for setSender
```

```
     *
     * @return void
     */
    public function testCanSetSender ()
    {
        $this->_header->setSender($this->_sender);

        $this->assertEquals($this->_header->getSender(), $this->
            _sender);
    }

    /**
     * Test for setRecipient
     *
     * @return void
     */
    public function testCanSetRecipient ()
    {
        $this->_header->setRecipient($this->_recipient);

        $this->assertEquals($this->_header->getRecipient(),
            $this->_recipient);
    }

    /**
     * Test for setMessageTime
     *
     * @return void
     */
    public function testCanSetMessageTimeAsString ()
    {
        $expectedDate = Zend_Date::now();
        $this->_header->setMessageTime($expectedDate->get());

        $actualDate = $this->_header->getMessageTime();
        $this->assertEquals($expectedDate, $actualDate);
    }

    /**
     * Test for setMessageTime
     *
     * @return void
     */
    public function testCanSetMessageTimeAsZendDate ()
    {
        $expectedDate = Zend_Date::now();
        $this->_header->setMessageTime($expectedDate);

        $actualDate = $this->_header->getMessageTime();
        $this->assertEquals($expectedDate, $actualDate);
    }
```

```
    /**
     * Test for setSenderKeyId
     *
     * @return void
     */
    public function testCanSetSenderKeyId ()
    {
        $senderKeyId = 'ABCD1234';
        $this->_header->setSenderKeyId($senderKeyId);

        $this->assertEquals($senderKeyId, $this->_header->
            getSenderKeyId());
    }

    /**
     * Test for setSenderKeyId
     *
     * @return void
     */
    public function testSetSenderKeyIdWithInvalidId ()
    {
        $this->setExpectedException('
            SGCI_Exception_InvalidParameter');

        $senderKeyId = array('noId');
        $this->_header->setSenderKeyId($senderKeyId);
    }

    /**
     * Test for setRecipientKeyId
     *
     * @return void
     */
    public function testCanSetRecipientKeyId ()
    {
        $recipientKeyId = 'ABCD1234';
        $this->_header->setRecipientKeyId($recipientKeyId);

        $this->assertEquals($recipientKeyId, $this->_header->
            getRecipientKeyId());
    }

    /**
     * Test for setRecipientKeyId
     *
     * @return void
     */
    public function testSetRecipientKeyIdWithInvalidId ()
    {
        $this->setExpectedException('
            SGCI_Exception_InvalidParameter');
```

```
        $recipientKeyId = new SGCI_PKIMessage();
        $this->_header->setRecipientKeyId($recipientKeyId);
    }

    /**
     * Test for setTransactionId
     *
     * @return void
     */
    public function testCanSetTransactionId ()
    {
        $transactionId = 0;
        $this->_header->setTransactionId($transactionId);

        $this->assertEquals($transactionId, $this->_header->
            getTransactionId());
    }

    /**
     * Test for setTransactionId
     *
     * @return void
     */
    public function testSetTransactionIdWithInvalidID ()
    {
        $this->setExpectedException('
            SGCI_Exception_InvalidParameter');

        $this->_header->setTransactionId(array('invalidID'));
    }

    /**
     * Test for setSenderNonce
     *
     * @return void
     */
    public function testCanSetSernderNonce ()
    {
        $nonce = 'ABC123';
        $this->_header->setSenderNonce($nonce);

        $this->assertEquals($nonce, $this->_header->
            getSenderNonce());
    }

    /**
     * Test for setSenderNonce
     *
     * @return void
     */
    public function testSetSenderNonceWithInvalidNonce ()
    {
```

```
        $this->setExpectedException('
            SGCI_Exception_InvalidParameter');

        $this->_header->setSenderNonce(array('invalidNonce'));
    }

    /**
     * Test for setRecipientNonce
     *
     * @return void
     */
    public function testCanSetRecipientNonce ()
    {
        $nonce = 'ABC123';
        $this->_header->setRecipientNonce($nonce);

        $this->assertEquals($nonce, $this->_header->
            getRecipientNonce());
    }

    /**
     * Test for setRecipientNonce
     *
     * @return void
     */
    public function testSetRecipientNonceWithInvalidNonce ()
    {
        $this->setExpectedException('
            SGCI_Exception_InvalidParameter');

        $this->_header->setRecipientNonce(Zend_Date::now());
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithoutSender ()
    {
        $this->setExpectedException('SGCI_Exception');

        $this->_header->setRecipient($this->_recipient);
        $this->_header->getXMLEncoded();
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithoutRecipient ()
```

```
    {
        $this->setExpectedException('SGCI_Exception');

        $this->_header->setSender($this->_sender);
        $this->_header->getXMLEncoded();
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedOnlyWithRequiredFields()
    {
        $expectedXML = $this->_getXML($this->_sender, $this->
            _recipient);

        $this->_setRequiredFields();

        $headerXML = $this->_header->getXMLEncoded();

        $xml = new DomDocument();
        $headerXML = $xml->importNode($headerXML, true);
        $xml->appendChild($headerXML);

        $validSchema = $xml->schemaValidateSource($this->
            _getPKIHeaderXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithOnlyMessageTimeSetted
        ()
    {
        $time = new Zend_Date();

        $messageTime = $time->getTimestamp();
        $expectedXML = $this->_getXML($this->_sender, $this->
            _recipient, $messageTime);

        $this->_setRequiredFields();

        $this->_header->setMessageTime($time);
        $headerXML = $this->_header->getXMLEncoded();

        $xml = new DomDocument();
```

```
        $headerXML = $xml->importNode($headerXML, true);
        $xml->appendChild($headerXML);

        $validSchema = $xml->schemaValidateSource($this->
            _getPKIHeaderXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithOnlySenderKIDSetted()
    {
        $senderKID = 'ABCD1234';
        $expectedXML = $this->_getXML(
            $this->_sender, $this->_recipient, null, null,
                $senderKID
        );

        $this->_setRequiredFields();
        $this->_header->setSenderKeyId($senderKID);

        $headerXML = $this->_header->getXMLEncoded();
        $xml = new DomDocument();
        $headerXML = $xml->importNode($headerXML, true);
        $xml->appendChild($headerXML);

        $validSchema = $xml->schemaValidateSource($this->
            _getPKIHeaderXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithOnlyRecipKIDSetted()
    {
        $recipKID = 'ABCD1234';
        $expectedXML = $this->_getXML(
            $this->_sender, $this->_recipient, null, null, null,
                $recipKID
        );

        $this->_setRequiredFields();
```

```
        $this->_header->setRecipientKeyId($recipKID);

        $headerXML = $this->_header->getXMLEncoded();
        $xml = new DomDocument();
        $headerXML = $xml->importNode($headerXML, true);
        $xml->appendChild($headerXML);

        $validSchema = $xml->schemaValidateSource($this->
            _getPKIHeaderXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithOnlyTransactionIDSetted
        ()
    {
        $transactionID = '2';
        $expectedXML = $this->_getXML(
            $this->_sender, $this->_recipient, null, null, null,
                null, $transactionID
        );

        $this->_setRequiredFields();
        $this->_header->setTransactionId($transactionID);

        $headerXML = $this->_header->getXMLEncoded();
        $xml = new DomDocument();
        $headerXML = $xml->importNode($headerXML, true);
        $xml->appendChild($headerXML);

        $validSchema = $xml->schemaValidateSource($this->
            _getPKIHeaderXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithOnlySenderNonceSetted
        ()
    {
        $senderNonce = 'ABCD1234';
```

```
        $expectedXML = $this->_getXML(
            $this->_sender, $this->_recipient, null, null, null,
                null, null, $senderNonce
        );

        $this->_setRequiredFields();
        $this->_header->setSenderNonce($senderNonce);

        $headerXML = $this->_header->getXMLEncoded();
        $xml = new DomDocument();
        $headerXML = $xml->importNode($headerXML, true);
        $xml->appendChild($headerXML);

        $validSchema = $xml->schemaValidateSource($this->
            _getPKIHeaderXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithOnlyRecipNonceSetted ()
    {
        $recipNonce = 'ABCD1234';
        $expectedXML = $this->_getXML(
            $this->_sender, $this->_recipient, null, null, null,
                null, null, null, $recipNonce
        );

        $this->_setRequiredFields();
        $this->_header->setRecipientNonce($recipNonce);

        $headerXML = $this->_header->getXMLEncoded();
        $xml = new DomDocument();
        $headerXML = $xml->importNode($headerXML, true);
        $xml->appendChild($headerXML);

        $validSchema = $xml->schemaValidateSource($this->
            _getPKIHeaderXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
```

```
     */
    public function testGetXMLEncodedWithAllValuesSetted ()
    {
        $this->_setRequiredFields();
        $recipNonce = 'Recp1234';
        $senderNonce = 'Sender1234';
        $transactionID = '4';
        $recipKID = 'RKID1234';
        $senderKID = 'SKID1234';
        $time = new Zend_Date();
        $messageTime = $time->getTimestamp();

        $protectionAlg =
            Labsec_Security_Certification_ObjectIdentifierFactory
                ::getObjectIdentifierByAlgorithm(
                Labsec_Security_Crypto_AsymmetricKey::
                    ALGORITHM_ECDSA,
                LabsecCL_Security_Crypto_MessageDigestCL::
                    ALGORITHM_SHA256
            );

        $expectedXML = $this->_getXML(
            $this->_sender, $this->_recipient,
            $messageTime, $protectionAlg, $senderKID, $recipKID,
            $transactionID, $senderNonce, $recipNonce
        );

        $this->_header->setMessageTime($time);
        $this->_header->setRecipientKeyId($recipKID);
        $this->_header->setRecipientNonce($recipNonce);
        $this->_header->setSenderKeyId($senderKID);
        $this->_header->setSenderNonce($senderNonce);
        $this->_header->setTransactionId($transactionID);
        $this->_header->setProtectionAlg($protectionAlg);

        $headerXML = $this->_header->getXMLEncoded();
        $xml = new DomDocument();
        $headerXML = $xml->importNode($headerXML, true);
        $xml->appendChild($headerXML);

        $validSchema = $xml->schemaValidateSource($this->
            _getPKIHeaderXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Sets the sender and the recipient of the PKIHeader
     *
     * @return void
     */
```

```
protected function _setRequiredFields ()
{
    $this->_header->setSender($this->_sender);
    $this->_header->setRecipient($this->_recipient);
}

/**
 * Returns a XML template for the PKIHeader structure
 *
 * @param
    Labsec_Security_Certification_DataTypes_DirectoryName
    $sender          The sender of
 *
                                                        the
    message
 * @param
    Labsec_Security_Certification_DataTypes_DirectoryName
    $recipient       The recipient
 *
                                                        of the
     message
 * @param string

    $messageTime     The time of
 *

    production of
 *
                                                        the
    message
 * @param string

    $protectionAlg   The protection
 *
                              algorithm used to
 *
                              sign the message
 * @param string

    $senderKID       The sender
 *
                                                        key
    identifier
 * @param string
```

```
         $recipientKID    The recipient
     *
                                                          key
         identifier
     * @param string
         $transactionId   The transaction
     *
         id
     * @param string
         $senderNonce     The nonce of
     *
         the sender
     * @param string
         $recipientNonce The nonce of
     *
         the recipient
     *
     * @return string
     */
    protected function _getXML
    (
        Labsec_Security_Certification_DataTypes_DirectoryName
            $sender,
        Labsec_Security_Certification_DataTypes_DirectoryName
            $recipient,
        $messageTime = null,
        $protectionAlg = null,
        $senderKID = null,
        $recipientKID = null,
        $transactionId = null,
        $senderNonce = null,
        $recipientNonce = null
    ) {
        $doc = new DOMDocument();

        $senderXML = $sender->getXMLEncoded('sender');
        $senderXML = $doc->importNode($senderXML, true);
        $senderXML = $doc->saveXML($senderXML);

        $recipientXML = $recipient->getXMLEncoded('recipient');
        $recipientXML = $doc->importNode($recipientXML, true);
        $recipientXML = $doc->saveXML($recipientXML);

        $messageTimeXML = '';
        if ($messageTime !== null) {
```

```
        $messageTimeXML = '<messageTime>' . $messageTime . '
            </messageTime>';
    }

    $protectionAlgXML = '';
    if ($protectionAlg !== null) {
        $protectionAlgXML = $protectionAlg->getXMLEncoded('
            protectionAlg');
        $protectionAlgXML = $doc->importNode(
            $protectionAlgXML, true);
        $protectionAlgXML = $doc->saveXML($protectionAlgXML)
            ;
    }

    $senderKIDXML = '';
    if ($senderKID !== null) {
        $senderKIDXML = '<senderKID>' . $senderKID . '</
            senderKID>';
    }

    $recipientKIDXML = '';
    if ($recipientKID !== null) {
        $recipientKIDXML = '<recipKID>' . $recipientKID . '
            </recipKID>';
    }

    $transactionIdXML = '';
    if ($transactionId !== null) {
        $transactionIdXML = '<transactionID>' .
            $transactionId . '</transactionID>';
    }

    $senderNonceXML = '';
    if ($senderNonce !== null) {
        $senderNonceXML = '<senderNonce>' . $senderNonce . '
            </senderNonce>';
    }

    $recipientNonceXML = '';
    if ($recipientNonce !== null) {
        $recipientNonceXML = '<recipNonce>' .
            $recipientNonce . '</recipNonce>';
    }

    return '<?xml version="1.0"?>' . PHP_EOL
        . '<PKIHeader>'
        . '<pvno>2</pvno>'
        . $senderXML
        . $recipientXML
        . $messageTimeXML
        . $protectionAlgXML
        . $senderKIDXML
```

```
            . $recipientKIDXML
            . $transactionIdXML
            . $senderNonceXML
            . $recipientNonceXML
            . '</PKIHeader>' . PHP_EOL;
    }

    /**
     * Returns the XML schema for the PKIHeader structure
     *
     * @return string
     */
    protected function _getPKIHeaderXMLSchema ()
    {
        return '<?xml version="1.0"?>
                        <xs:schema xmlns:xs="http://www.w3.org
                            /2001/XMLSchema">
                        <xs:element name="PKIHeader">
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element ref
                                                    ="pvno"/>
                                                <xs:element ref
                                                    ="sender"/>
                                                <xs:element ref
                                                    ="recipient
                                                    "/>
                                                <xs:element name
                                                    ="
                                                    messageTime
                                                    " type="xs:
                                                    integer"
                                                    minOccurs
                                                    ="0"/> <!--
                                                     Optional
                                                    -->
                                                <xs:element ref
                                                    ="
                                                    protectionAlg
                                                    " minOccurs
                                                    ="0"/> <!--
                                                     Optional
                                                    -->
                                                <xs:element name
                                                    ="senderKID
                                                    " type="xs:
                                                    string"
                                                    minOccurs
                                                    ="0"/> <!--
                                                     Optional
                                                    -->
                                                <xs:element name
```

```
                                        ="recipKID"
                                         type="xs:
                                        string"
                                        minOccurs
                                        ="0"/> <!--
                                         Optional
                                        -->
                                    <xs:element name
                                        ="
                                        transactionID
                                        " type="xs:
                                        string"
                                        minOccurs
                                        ="0"/> <!--
                                         Optional
                                        -->
                                    <xs:element name
                                        ="
                                        senderNonce
                                        " type="xs:
                                        string"
                                        minOccurs
                                        ="0"/> <!--
                                         Optional
                                        -->
                                    <xs:element name
                                        ="
                                        recipNonce"
                                         type="xs:
                                        string"
                                        minOccurs
                                        ="0"/> <!--
                                         Optional
                                        -->
                            </xs:sequence>
                    </xs:complexType>
            </xs:element>

            <xs:element name="pvno">
                    <xs:simpleType>
                            <xs:restriction base="xs
                                :integer">
                                <xs:enumeration
                                    value="2"/>
                        </xs:restriction>
                    </xs:simpleType>
            </xs:element>

            <xs:element name="sender"
                substitutionGroup="GeneralName"/>
            <xs:element name="recipient"
                substitutionGroup="GeneralName"/>
```

```
<xs:element name="GeneralName">
        <xs:complexType>
                <xs:choice>
                        <xs:element ref
                            ="otherName
                            "/>
                        <xs:element name
                            ="
                            rfc822Name"
                             type="xs:
                            string"/>
                        <xs:element name
                            ="dNSName"
                            type="xs:
                            string"/>
                        <xs:element ref
                            ="
                            directoryName
                            "/>
                        <xs:element name
                            ="
                            uniformResourceIdentifier
                            " type="xs:
                            string"/>
                        <xs:element name
                            ="iPAddress
                            " type="xs:
                            string"/>
                        <xs:element name
                            ="
                            registeredID
                            " type="xs:
                            string"/>
                </xs:choice>
        </xs:complexType>
</xs:element>

<xs:element name="protectionAlg"
    substitutionGroup="
    AlgorithmIdentifier"/>

<xs:element name="AlgorithmIdentifier">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name
                            ="algorithm
                            " type="xs:
                            string"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

```
<xs:element name="otherName"
    substitutionGroup="OtherName"/>

<xs:element name="OtherName">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name
                            ="type-id"
                            type="xs:
                            string"/>
                        <xs:element name
                            ="value"
                            type="xs:
                            string"/>
                            <!--
                            EXPLICIT
                            ANY DEFINED
                             BY type-id
                             -->
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="directoryName"
    substitutionGroup="RDNSequence"/>

<xs:element name="RDNSequence">
        <xs:complexType>
                <xs:sequence>
                        <xs:choice>
                                <xs:
                                    element

                                    name
                                    ="C
                                    "
                                    type
                                    ="
                                    xs:
                                    string
                                    "/>

                                    <!--

                                    Country

                                    -->
                                <xs:
                                    element

                                    name
                                    ="L
```

```
                                                                    "
                                                                    type
                                                                    ="
                                                                    xs:
                                                                    string
                                                                    "/>

                                                                    <!--

                                                                    Locality

                                                                    -->
                                                                <xs:
                                                                    element

                                                                    name
                                                                    ="
                                                                    ST"

                                                                    type
                                                                    ="
                                                                    xs:
                                                                    string
                                                                    "/>

                                                                    <!--

                                                                    State
                                                                     or

                                                                    province

                                                                    -->
                                                                <xs:
                                                                    element

                                                                    name
                                                                    ="O
                                                                    "
                                                                    type
                                                                    ="
                                                                    xs:
                                                                    string
                                                                    "/>

                                                                    <!--

                                                                    Organization

                                                                    -->
                                                                <xs:
                                                                    element
```

```
                name
                ="
                OU"

                type
                ="
                xs:
                string
                "/>

                <!--

                Organization

                Unit

                -->
            <xs:
                element

                name
                ="
                CN"

                type
                ="
                xs:
                string
                "/>

                <!--

                Common

                Name

                -->
            <xs:
                element

                name
                ="
                STREET
                "
                type
                ="
                xs:
                string
                "/>

                <!--
```

```
                                                                    Street

                                                                    Address

                                                                    -->
                                                                <xs:
                                                                    element

                                                                    name
                                                                    ="E
                                                                    "
                                                                    type
                                                                    ="
                                                                    xs:
                                                                    string
                                                                    "/>

                                                                    <!--
                                                                     E-
                                                                    mail

                                                                    Address

                                                                    -->
                                                            </xs:choice>
                                                    </xs:sequence>
                                            </xs:complexType>
                                    </xs:element>
                                    </xs:schema>';
    }
}
```

Código Fonte C.43: Classe SGCI_PKIMessage_HeaderTest

```
<?php
/**
 * Class that tests the
     SGCI_PKIMessage_CertificationRequestMessage class
 */
class SGCI_PKIMessage_Body_CertificationRequestMessageTest
    extends PHPUnit_Framework_TestCase
{
    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncoded ()
    {
        $body = self::getCertificationRequestMessage();
```

```
        $expectedXML =
            SGCI_PKIMessage_Body_CertificationRequestMessagesTest
            ::getXML(
            $body->getCertReqMsgs(), 'cr'
        );

        $certReqXML = $body->getXMLEncoded();

        $xml = new DomDocument();

        $certReqXML = $xml->importNode($certReqXML, true);
        $xml->appendChild($certReqXML);

        $validSchema = $xml->schemaValidateSource(
            SGCI_PKIMessage_Body_CertificationRequestMessagesTest
                ::getCertReqMsgXMLSchema()
        );

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Returns a
          SGCI_PKIMessage_Body_CertificationRequestMessage
          object
     *
     * @return SGCI_PKIMessage_Body_CertificationRequestMessage
     */
    public static function getCertificationRequestMessage()
    {
        $certTemplate = new SGCI_PKIMessage_CertificateTemplate
            ();
        $certTemplate->setSerialNumber(10);

        $certRequest = new
            SGCI_PKIMessage_CertificationRequestMessage();
        $certRequest->setCertReqId(3);
        $certRequest->setCertTemplate($certTemplate);

        $cr = new
            SGCI_PKIMessage_Body_CertificationRequestMessage();
        $cr->addCertReqMsg($certRequest);

        return $cr;
    }
}
```

Código Fonte C.44: Classe SGCI_PKIMessage_Body_CertificationRequestMessageTest

```
<?php
/**
 * Class that tests the
     SGCI_PKIMessage_Body_CertificationRequestMessages class
 */
class SGCI_PKIMessage_Body_CertificationRequestMessagesTest
    extends PHPUnit_Framework_TestCase
{
    /**
     * Class being tested.
     *
     * @var SGCI_PKIMessage_Body_CertificationRequestMessages
     */
    protected $_certRequests;

    /**
     * Instantiate a
         SGCI_PKIMessage_Body_CertificationRequestMessages
         object
     *
     * @return void
     */
    public function setUp ()
    {
        $this->_certRequests = new
            SGCI_PKIMessage_Body_CertificationRequestMessages()
            ;
    }

    /**
     * Test for loadXML
     *
     * @return void
     */
    public function testLoadXMLWithOneCertReq ()
    {
        $certTemplate = new SGCI_PKIMessage_CertificateTemplate
            ();
        $certTemplate->setSerialNumber(10);

        $certRequest = new
            SGCI_PKIMessage_CertificationRequestMessage();
        $certRequest->setCertReqId(3);
        $certRequest->setCertTemplate($certTemplate);

        $xml = $this->_getDomNode(array($certRequest));

        $this->_certRequests->loadXML($xml);

        $certReqMsgs = $this->_certRequests->getCertReqMsgs();

        $this->assertEquals(1, count($certReqMsgs));
```

```
        $this->assertEquals($certRequest, $certReqMsgs[0]);
    }

    /**
     * Test for loadXML
     *
     * @return void
     */
    public function testLoadXMLWithTwoCertReqs()
    {
        $certReqMsgs = array();

        $certTemplate = new SGCI_PKIMessage_CertificateTemplate
            ();
        $certTemplate->setSerialNumber(2);

        $certRequestA = new
            SGCI_PKIMessage_CertificationRequestMessage();
        $certRequestA->setCertReqId(1);
        $certRequestA->setCertTemplate($certTemplate);
        $certReqMsgs[] = $certRequestA;

        $certTemplate = new SGCI_PKIMessage_CertificateTemplate
            ();
        $certTemplate->setSerialNumber(5);
        $certTemplate->setVersion(0);

        $certRequestB = new
            SGCI_PKIMessage_CertificationRequestMessage();
        $certRequestB->setCertReqId(3);
        $certRequestB->setCertTemplate($certTemplate);
        $certReqMsgs[] = $certRequestB;

        $xml = $this->_getDomNode($certReqMsgs);

        $this->_certRequests->loadXML($xml);

        $certReqMsgs = $this->_certRequests->getCertReqMsgs();

        $this->assertEquals(2, count($certReqMsgs));
        $this->assertEquals($certRequestA, $certReqMsgs[0]);
        $this->assertEquals($certRequestB, $certReqMsgs[1]);
    }

    /**
     * Construct the XML from the parameters and return a
         DomNode representation of the XML
     *
     * @param array $certReqMsgs The requests of the message
     *
     * @return void
     */
```

```
    protected function _getDomNode (array $certReqMsgs)
    {
        $xml = self::getXML($certReqMsgs);

        $dom = new DOMDocument();
        $dom->loadXML($xml);

        return $dom->getElementsByTagName('CertReqMessages')->
            item(0);
    }

    /**
     * Test for addCertReqMsg
     *
     * @return void
     */
    public function testCanAddCertReq ()
    {
        $certReq = new
            SGCI_PKIMessage_CertificationRequestMessage();

        $this->_certRequests->addCertReqMsg($certReq);

        $certReqs = $this->_certRequests->getCertReqMsgs();

        $this->assertEquals(1, count($certReqs));
        $this->assertEquals($certReq, $certReqs[0]);
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithoutAddingACertReq ()
    {
        $this->setExpectedException('SGCI_Exception');

        $this->_certRequests->getXMLEncoded();
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithOneCertReq ()
    {
        $certTemplate = new SGCI_PKIMessage_CertificateTemplate
            ();
        $certTemplate->setSerialNumber(10);
```

```
        $certRequest = new
            SGCI_PKIMessage_CertificationRequestMessage();
        $certRequest->setCertReqId(3);
        $certRequest->setCertTemplate($certTemplate);
        $this->_certRequests->addCertReqMsg($certRequest);

        $expectedXML = self::getXML(array($certRequest));

        $certReqXML = $this->_certRequests->getXMLEncoded();

        $xml = new DomDocument();

        $certReqXML = $xml->importNode($certReqXML, true);
        $xml->appendChild($certReqXML);

        $validSchema = $xml->schemaValidateSource(self::
            getCertReqMsgXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithTwoCertReq ()
    {
        $certReqMsgs = array();

        $certTemplate = new SGCI_PKIMessage_CertificateTemplate
            ();
        $certTemplate->setSerialNumber(2);

        $certRequest = new
            SGCI_PKIMessage_CertificationRequestMessage();
        $certRequest->setCertReqId(1);
        $certRequest->setCertTemplate($certTemplate);
        $this->_certRequests->addCertReqMsg($certRequest);
        $certReqMsgs[] = $certRequest;

        $certTemplate = new SGCI_PKIMessage_CertificateTemplate
            ();
        $certTemplate->setSerialNumber(5);
        $certTemplate->setVersion(0);

        $certRequest = new
            SGCI_PKIMessage_CertificationRequestMessage();
        $certRequest->setCertReqId(3);
        $certRequest->setCertTemplate($certTemplate);
        $this->_certRequests->addCertReqMsg($certRequest);
```

```
        $certReqMsgs[] = $certRequest;

        $expectedXML = self::getXML($certReqMsgs);
        $certReqXML = $this->_certRequests->getXMLEncoded();

        $xml = new DomDocument();

        $certReqXML = $xml->importNode($certReqXML, true);
        $xml->appendChild($certReqXML);
        $validSchema = $xml->schemaValidateSource(self::
            getCertReqMsgXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Construct the XML from the parameters and return a string
          representation of the XML
     *
     * @param array  $certReqMsgs The requests of the message
     * @param string $nodeName    The name of the root node of
        the generated XML
     *
     * @return void
     */
    public static function getXML (array $certReqMsgs, $nodeName
        = 'CertReqMessages')
    {
        $certReqMsgsXML = '';
        foreach ($certReqMsgs as $certReqMsg) {
            $certReqMsgXML = $certReqMsg->getXMLEncoded('
                CertReqMsg');
            $doc = new DOMDocument();
            $certReqMsgXML = $doc->importNode($certReqMsgXML,
                true);
            $certReqMsgXML = $doc->saveXML($certReqMsgXML);
            $certReqMsgsXML .= $certReqMsgXML;
        }

        $xml = '<?xml version="1.0"?>' . PHP_EOL
            . '<' . $nodeName . '>'
            . $certReqMsgsXML
            . '</' . $nodeName . '>' . PHP_EOL;

        return $xml;
    }

    /**
     * Returns the XML schema for the CertReqMessages structure
     *
     * @return string
```

```
 */
public static function getCertReqMsgXMLSchema ()
{
    return '<?xml version="1.0"?>
                    <xs:schema xmlns:xs="http://www.w3.org
                        /2001/XMLSchema">

                    <xs:element name="cr" substitutionGroup
                        ="CertReqMessages"/>
                    <xs:element name="ir" substitutionGroup
                        ="CertReqMessages"/>

                    <xs:element name="CertReqMessages">
                            <xs:complexType>
                                    <xs:sequence>
                                            <xs:element ref
                                                ="
                                                CertReqMsg"
                                                 maxOccurs
                                                ="unbounded
                                                "/>
                                    </xs:sequence>
                            </xs:complexType>
                    </xs:element>

                    <xs:element name="CertReqMsg">
                            <xs:complexType>
                                    <xs:sequence>
                                            <xs:element ref
                                                ="certReq
                                                "/>
                                    </xs:sequence>
                            </xs:complexType>
                    </xs:element>

                    <xs:element name="certReq"
                        substitutionGroup="CertRequest"/>

                    <xs:element name="CertRequest">
                            <xs:complexType>
                                    <xs:sequence>
                                            <xs:element name
                                                ="certReqId
                                                " type="xs:
                                                integer"/>
                                            <xs:element ref
                                                ="
                                                certTemplate
                                                "/>
                                    </xs:sequence>
                            </xs:complexType>
                    </xs:element>
```

```
<xs:element name="certTemplate"
    substitutionGroup="CertTemplate"/>

<xs:element name="CertTemplate">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref
                            ="version"
                            minOccurs
                            ="0"/> <!--
                             Optional
                            -->
                        <xs:element name
                            ="
                            serialNumber
                            " minOccurs
                            ="0" type="
                            xs:integer
                            "/> <!--
                            Optional
                            -->
                        <xs:element ref
                            ="subject"
                            minOccurs
                            ="0"/> <!--
                             Optional
                            -->
                        <xs:element name
                            ="publicKey
                            " minOccurs
                            ="0" type="
                            xs:string
                            "/> <!--
                            Optional
                            -->
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="version"
    substitutionGroup="Version"/>

<xs:element name="Version">
        <xs:simpleType>
                <xs:restriction base="xs
                    :integer">
                        <xs:enumeration
                            value="0"/>
                             <!-- v1
                            -->
                        <xs:enumeration
```

```
                                        value="1"/>
                                         <!-- v2
                                        -->
                                    <xs:enumeration
                                        value="2"/>
                                         <!-- v3
                                        -->
                            </xs:restriction>
                        </xs:simpleType>
                </xs:element>

                <xs:element name="subject"
                    substitutionGroup="RDNSequence"/>

                <xs:element name="RDNSequence">
                        <xs:complexType>
                                <xs:sequence>
                                        <xs:choice>
                                                <xs:
                                                    element

                                                    name
                                                    ="C
                                                    "
                                                    type
                                                    ="
                                                    xs:
                                                    string
                                                    "/>

                                                    <!--

                                                    Country

                                                    -->
                                                <xs:
                                                    element

                                                    name
                                                    ="L
                                                    "
                                                    type
                                                    ="
                                                    xs:
                                                    string
                                                    "/>

                                                    <!--

                                                    Locality

                                                    -->
```

```
                                                    <xs:
                                                        element

                                                        name
                                                        ="
                                                        ST"

                                                        type
                                                        ="
                                                        xs:
                                                        string
                                                        "/>

                                                        <!--

                                                        State
                                                         or

                                                        province

                                                        -->
                                                    <xs:
                                                        element

                                                        name
                                                        ="O
                                                        "
                                                        type
                                                        ="
                                                        xs:
                                                        string
                                                        "/>

                                                        <!--

                                                        Organization

                                                        -->
                                                    <xs:
                                                        element

                                                        name
                                                        ="
                                                        OU"

                                                        type
                                                        ="
                                                        xs:
                                                        string
                                                        "/>

                                                        <!--
```

```
                                    Organization
                                    Unit
                                    -->
                                <xs:
                                    element
                                    name
                                    ="
                                    CN"
                                    type
                                    ="
                                    xs:
                                    string
                                    "/>
                                    <!--
                                    Common
                                    Name
                                    -->
                                <xs:
                                    element
                                    name
                                    ="
                                    STREET
                                    "
                                    type
                                    ="
                                    xs:
                                    string
                                    "/>
                                    <!--
                                    Street
                                    Address
                                    -->
                                <xs:
                                    element
                                    name
                                    ="E
                                    "
```

```
                                                                    type
                                                                    ="
                                                                    xs:
                                                                    string
                                                                    "/>

                                                                    <!--
                                                                     E-
                                                                    mail

                                                                    Address

                                                                    -->
                                                        </xs:choice>
                                                </xs:sequence>
                                        </xs:complexType>
                                </xs:element>
                                </xs:schema>';
    }
}
```

Código Fonte C.45: Classe SGCI_PKIMessage_Body_CertificationRequestMessagesTest

```
<?php
/**
 * Class that tests the
     SGCI_PKIMessage_Body_CertificationResponse class
 */
class SGCI_PKIMessage_Body_CertificationResponseTest extends
    PHPUnit_Framework_TestCase
{
    /**
     * Class being tested.
     *
     * @var SGCI_PKIMessage_Body_CertificationResponse
     */
    protected $_certRespMessage;

    /**
     * Instantiate a SGCI_PKIMessage_Body_CertificationResponse
         object.
     *
     * @return void
     */
    public function setUp ()
    {
        $this->_certRespMessage = new
            SGCI_PKIMessage_Body_CertificationResponse();
    }
```

```
/**
 * Test for loadXML.
 *
 * @return void
 */
public function testLoadXMLWithOnlyRequiredFields ()
{
    $certReqId = 1;
    $expectedStatus = SGCI_PKIStatus::STATUS_WAITING;

    $pkiStatusInfo = new SGCI_PKIStatusInfo();
    $pkiStatusInfo->setStatus($expectedStatus);

    $xml = $this->_getDomNode($certReqId, $pkiStatusInfo);

    $this->_certRespMessage->loadXML($xml);

    $this->assertEquals($certReqId, $this->_certRespMessage
        ->getCertReqId());

    $status = $this->_certRespMessage->getStatus()->
        getStatus();
    $this->assertEquals($expectedStatus, $status);
    $this->assertEquals(0, count($this->_certRespMessage->
        getCaPubs()));
    $this->assertNull($this->_certRespMessage->
        getCertificate());
}

/**
 * Test for loadXML.
 *
 * @return void
 */
public function testLoadXMLWithAllFields ()
{
    $certReqId = 1;
    $expectedStatus = SGCI_PKIStatus::
        STATUS_GRANTED_WITH_MODS;

    $caPubCert = $this->_getCertificate();
    $certificate = $this->_getCertificate();

    $pkiStatusInfo = new SGCI_PKIStatusInfo();
    $pkiStatusInfo->setStatus($expectedStatus);

    $xml = $this->_getDomNode($certReqId, $pkiStatusInfo,
        array($caPubCert), $certificate);

    $this->_certRespMessage->loadXML($xml);

    $this->assertEquals($certReqId, $this->_certRespMessage
```

```
            ->getCertReqId());

        $status = $this->_certRespMessage->getStatus()->
            getStatus();
        $this->assertEquals($expectedStatus, $status);

        $caPubs = $this->_certRespMessage->getCaPubs();
        $this->assertEquals(1, count($caPubs));
        $this->assertEquals($caPubCert->getPem(), $caPubs[0]->
            getPem());

        $certPem = $this->_certRespMessage->getCertificate()->
            getPem();
        $this->assertEquals($certificate->getPem(), $certPem);
    }

    /**
     * Construct the XML from the parameters and return a
         DomNode representation of the XML.
     *
     * @param integer              $certReqId The id of the
         request.
     * @param SGCI_PKIStatusInfo $status    The status of the
         response.
     * @param array               $caPubs    Extra certs.
     * @param LabsecCL_Security_Certification_CertificateCL
         $certificate The certificate issued.
     *
     * @return DOMNode
     */
    protected function _getDomNode
    (
        $certReqId,
        SGCI_PKIStatusInfo $status,
        array $caPubs = array(),
        $certificate = null
    ) {
        $xml = self::getXML($certReqId, $status, $caPubs,
            $certificate);

        $dom = new DOMDocument();
        $dom->loadXML($xml);

        return $dom->getElementsByTagName('CertRepMessage')->
            item(0);
    }

    /**
     * Test for AddCaPub.
     *
     * @return void
     */
```

```
    public function testCanAddCaPub ()
    {
        $certificate = $this->_getCertificate();

        $this->_certRespMessage->addCaPub($certificate);
        $caPubs = $this->_certRespMessage->getCaPubs();
        $actualCertificate = $caPubs[0];

        $this->assertEquals(count($caPubs), 1);
        $this->assertEquals($certificate, $actualCertificate);
    }

    /**
     * Test for AddCaPub.
     *
     * @return void
     */
    public function testCanAddTwoCaPubs ()
    {
        $firstCertificate = $this->_getCertificate();
        $secondCertificate = $this->_getCertificate();

        $this->_certRespMessage->addCaPub($firstCertificate);
        $this->_certRespMessage->addCaPub($secondCertificate);

        $caPubs = $this->_certRespMessage->getCaPubs();

        $this->assertEquals(count($caPubs), 2);
    }

    /**
     * Test for SetReqId.
     *
     * @return void
     */
    public function testCanSetCertReqId ()
    {
        $expectedCertReqId = 2;
        $this->_certRespMessage->setCertReqId($expectedCertReqId
            );

        $actualCertReqId = $this->_certRespMessage->getCertReqId
            ();

        $this->assertEquals($expectedCertReqId, $actualCertReqId
            );
    }

    /**
     * Test for setCertReqId.
     *
     * @param mixed $certReqId Id provided by the
```

```
     invalidCertReqIdProvider.
     *
     * @dataProvider invalidCertReqIdProvider
     * @return void
     */
    public function testSetCertReqIdWithInvalidId ($certReqId)
    {
        $this->setExpectedException('
            SGCI_Exception_InvalidParameter');

        $this->_certRespMessage->setCertReqId($certReqId);
    }

    /**
     * DataProvider with invalid certReqIds.
     *
     * @return mixed
     */
    public function invalidCertReqIdProvider ()
    {
        return array(
                array('invalid'),
                array(array('invalid')),
               );
    }

    /**
     * Test for setStatus.
     *
     * @return void
     */
    public function testCanSetStatus ()
    {
        $status = new SGCI_PKIStatusInfo();
        $status->setStatus(SGCI_PKIStatus::STATUS_ACCEPTED);
        $this->_certRespMessage->setStatus($status);

        $actualStatus = $this->_certRespMessage->getStatus();

        $this->assertEquals($status, $actualStatus);
    }

    /**
     * Test for setStatus.
     *
     * @return void
     */
    public function testSetStatusWithEmptyStatus ()
    {
        $this->setExpectedException('
            SGCI_Exception_InvalidParameter');
```

```
        $status = new SGCI_PKIStatusInfo();
        $this->_certRespMessage->setStatus($status);
    }

    /**
     * Test for SetCertificate.
     *
     * @return void
     */
    public function
        testSetCertificateWithoutStatusSetShouldThrowException
        ()
    {
        $this->setExpectedException('SGCI_Exception');

        $this->_certRespMessage->setCertificate($this->
            _getCertificate());
    }

    /**
     * Test for SetCertificate.
     *
     * @return void
     */
    public function
        testSetCertificateWithNotApprovalOrGrantedWithModsStatusShouldThr
         ()
    {
        $this->setExpectedException('SGCI_Exception');

        $status = new SGCI_PKIStatusInfo();
        $status->setStatus(SGCI_PKIStatus::STATUS_REJECTION);
        $this->_certRespMessage->setStatus($status);
        $this->_certRespMessage->setCertificate($this->
            _getCertificate());
    }

    /**
     * Test for SetCertificate.
     *
     * @return void
     */
    public function testCanSetCertificate ()
    {
        $status = new SGCI_PKIStatusInfo();
        $status->setStatus(SGCI_PKIStatus::STATUS_ACCEPTED);
        $this->_certRespMessage->setStatus($status);

        $certificate = $this->_getCertificate();
        $this->_certRespMessage->setCertificate($certificate);

        $actualCertificate = $this->_certRespMessage->
```

```
            getCertificate();

        $this->assertEquals($certificate, $actualCertificate);
    }

    /**
     * Test for GetXMLEncoded.
     *
     * @return void
     */
    public function
        testGetXMLEncodedWithoutStatusShouldThrowException()
    {
        $this->setExpectedException('SGCI_Exception');

        $this->_certRespMessage->setCertReqId(1);
        $this->_certRespMessage->getXMLEncoded();
    }

    /**
     * Test for GetXMLEncoded.
     *
     * @return void
     */
    public function
        testGetXMLEncodedWithoutCertReqIdShouldThrowException
        ()
    {
        $this->setExpectedException('SGCI_Exception');

        $status = new SGCI_PKIStatusInfo();
        $status->setStatus(0);
        $this->_certRespMessage->setStatus($status);
        $this->_certRespMessage->getXMLEncoded();
    }

    /**
     * Test for GetXMLEncoded.
     *
     * @return void
     */
    public function testGetXMLEncodedWithOnlyRequiredFields()
    {
        $status = SGCI_PKIStatus::STATUS_ACCEPTED;
        $pkiStatusInfo = new SGCI_PKIStatusInfo();
        $pkiStatusInfo->setStatus($status);

        $certReqId = 1;
        $expectedXML = self::getXML($certReqId, $pkiStatusInfo);
        $this->_certRespMessage->setCertReqId($certReqId);
        $this->_certRespMessage->setStatus($pkiStatusInfo);
```

```
        $certRepXML = $this->_certRespMessage->getXMLEncoded();

        $xml = new DomDocument();
        $certRepXML = $xml->importNode($certRepXML, true);
        $xml->appendChild($certRepXML);

        $validSchema = $xml->schemaValidateSource(self::
            getCertRepMsgXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Test for GetXMLEncoded.
     *
     * @return void
     */
    public function testGetXMLEncodedWithAllFields ()
    {
        $caPubCert = $this->_getCertificate();
        $certificate = $this->_getCertificate();
        $pkiStatusInfo = new SGCI_PKIStatusInfo();
        $pkiStatusInfo->setStatus(SGCI_PKIStatus::
            STATUS_ACCEPTED);

        $certReqId = 1;
        $expectedXML = self::getXML($certReqId, $pkiStatusInfo,
            array($caPubCert), $certificate);

        $this->_certRespMessage->setCertReqId($certReqId);
        $this->_certRespMessage->setStatus($pkiStatusInfo);
        $this->_certRespMessage->setCertificate($certificate);
        $this->_certRespMessage->addCaPub($caPubCert);

        $certRepXML = $this->_certRespMessage->getXMLEncoded();

        $xml = new DomDocument();
        $certRepXML = $xml->importNode($certRepXML, true);
        $xml->appendChild($certRepXML);

        $validSchema = $xml->schemaValidateSource(self::
            getCertRepMsgXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Construct the XML from the parameters and return a string
          representation of the XML.
     *
```

```
     * @param integer              $certReqId The id of the
         request.
     * @param SGCI_PKIStatusInfo $status    The status of the
         response.
     * @param array                $caPubs    Extra certs.
     * @param LabsecCL_Security_Certification_CertificateCL
         $certificate The certificate issued.
     * @param string $nodeName The name of the root node of the
         generated XML.
     *
     * @throws Exception
     * @return string
     */
    public static function getXML (
        $certReqId,
        SGCI_PKIStatusInfo $status,
        array $caPubs = array(),
        $certificate = null,
        $nodeName = 'CertRepMessage'
    ) {
        $caPubsXML = '';
        if (empty($caPubs) === false) {
            foreach ($caPubs as $caPub) {
                if ($caPub instanceof
                    LabsecCL_Security_Certification_CertificateCL
                    ) {
                    $certificatePem = $caPub->getPem();
                    $caPubsXML .= '<certificate>' .
                        $certificatePem . '</certificate>';
                } else {
                    throw new Exception(
                        'Each CaPub must be an instance of' . '
                            '
                        . '
                            LabsecCL_Security_Certification_CertificateCL
                            '
                    );
                }
            }
            $caPubsXML = '<caPubs>' . $caPubsXML . '</caPubs>';
        }

        $certReqIdXML = '<certReqId>' . $certReqId . '</
            certReqId>';

        $doc = new DOMDocument();
        $statusXML = $status->getXMLEncoded('status');
        $statusXML = $doc->importNode($statusXML, true);
        $statusXML = $doc->saveXML($statusXML);

        $certificateXML = '';
        if ($certificate !== null) {
```

```
            $certificatePem = $certificate->getPem();
            $certificateXML = '<certifiedKeyPair>' . '<
                certOrEncCert>'
                            . '<certificate>' . $certificatePem
                                . '</certificate>'
                            . '</certOrEncCert>' . '</
                                certifiedKeyPair>';
        }

        $xml = '<?xml version="1.0"?>' . PHP_EOL
             . '<' . $nodeName . '>'
             . $caPubsXML
             . '<response>'
             . $certReqIdXML
             . $statusXML
             . $certificateXML
             . '</response>'
             . '</' . $nodeName . '>' . PHP_EOL;

        return $xml;
    }

    /**
     * Returns a certificate object.
     *
     * @return LabsecCL_Security_Certification_CertificateCL
     */
    protected function _getCertificate ()
    {
        $data = '-----BEGIN CERTIFICATE-----' . PHP_EOL .
            '
                MIIB5TCCAU6gAwIBAgIBATANBgkqhkiG9w0BAQUFADAPMQ0wCwYDVQQDE
                ' . PHP_EOL
            . '
                MB4XDTEwMDgwNjE1MzAwMloXDTIwMDgwMzE1MzAwMlowDzENMAsGA1UEA
                ' . PHP_EOL
            . '
                dDCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAnp1d1wAcIhDFD00Iq
                ' . PHP_EOL
            . 'j+
                Jqtyf8aY19Xr8UYYFA98AyKrbQ9rSWu7Cun3C09yRLu4zbLDfALTUxqGV
                ' . PHP_EOL
            . 'baOK5HCpo63vNT+
                Ktkrd9Zt1cf86XWgVZI2iOwHX4jacx7QLALo2zdVmmpPDMydf
                ' . PHP_EOL
            . 'h/bg/Kwybe08K/EZO60CAwEAAaNRME8wDwYDVR0TAQH/
                BAUwAwEB/zAOBgNVHQ8B' . PHP_EOL
            . 'Af8EBAMCAQYwHQYDVR0OBBYEFJ8KY2/9nEEfNUep/bmBfHG9/
                iRjMA0GA1UdEQQG' . PHP_EOL
            . '
                MASkAjAAMA0GCSqGSIb3DQEBBQUAA4GBAH4LsmdDogNQu5Wej6rQGL8TG
                ' . PHP_EOL
```

```
            . 'TugTGQo/94GpS+6vna1bkNIotTmSdNLBPHTOuvBD+
                nL9Ku3tezjmgsPV+BmblEFE' . PHP_EOL
            . 'v58L1GnapdQXfJe+
                fNcpmYZ0TmXwE12TvvL6aWFMfnGniMcbBb4QQg1c10Y2dmjN
                ' . PHP_EOL
            . 'yHP8BB85fMxn' . PHP_EOL
            . '-----END CERTIFICATE-----';

        return new LabsecCL_Security_Certification_CertificateCL
            ($data);
    }

    /**
     * Returns the schema validator of the CertRepMessage
         structure.
     *
     * @return string
     */
    public static function getCertRepMsgXMLSchema ()
    {
        return '<?xml version="1.0"?>
                        <xs:schema xmlns:xs="http://www.w3.org
                            /2001/XMLSchema">

                        <xs:element name="cp" substitutionGroup
                            ="CertRepMessage"/>
                        <xs:element name="ip" substitutionGroup
                            ="CertRepMessage"/>

                        <xs:element name="CertRepMessage">
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element ref
                                                    ="caPubs"
                                                    minOccurs
                                                    ="0"/>
                                                <xs:element ref
                                                    ="response
                                                    "/>
                                        </xs:sequence>
                                </xs:complexType>
                        </xs:element>

                        <xs:element name="caPubs">
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element name
                                                    ="
                                                    certificate
                                                    " type="xs:
                                                    string"
                                                    maxOccurs="
```

```
                                    unbounded
                                    "/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>

        <xs:element name="response"
            substitutionGroup="CertResponse"/>

        <xs:element name="CertResponse">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name
                                    ="certReqId
                                    " type="xs:
                                    integer"/>
                                <xs:element ref
                                    ="status"/>
                                <xs:element ref
                                    ="
                                    certifiedKeyPair
                                    " minOccurs
                                    ="0"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>

        <xs:element name="status"
            substitutionGroup="PKIStatusInfo"/>

        <xs:element name="PKIStatusInfo">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name
                                    ="status"
                                    type="xs:
                                    integer"/>
                                <xs:element name
                                    ="
                                    statusString
                                    " type="xs:
                                    string"
                                    minOccurs
                                    ="0"/>
                                <xs:element name
                                    ="failInfo"
                                     type="xs:
                                    integer"
                                    minOccurs
                                    ="0"/>
                        </xs:sequence>
                </xs:complexType>
```

```
                        </xs:element>

                        <xs:element name="certifiedKeyPair"
                            substitutionGroup="CertifiedKeyPair
                            "/>

                        <xs:element name="CertifiedKeyPair">
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element ref
                                                    ="
                                                    certOrEncCert
                                                    "/>
                                        </xs:sequence>
                                </xs:complexType>
                        </xs:element>

                        <xs:element name="certOrEncCert">
                                <xs:complexType>
                                        <xs:choice>
                                                <xs:element name
                                                    ="
                                                    certificate
                                                    " type="xs:
                                                    string"/>
                                                <xs:element name
                                                    ="
                                                    encryptedCert
                                                    " type="xs:
                                                    string"/>
                                        </xs:choice>
                                </xs:complexType>
                        </xs:element>

                        </xs:schema>';
    }
}
```

Código Fonte C.46: Classe SGCI_PKIMessage_Body_CertificationResponseTest

```
<?php
/**
 * Class that tests the
     SGCI_PKIMessage_Body_CertificationResponseMessage class
 */
class SGCI_PKIMessage_Body_CertificationResponseMessageTest
    extends PHPUnit_Framework_TestCase
{
    /**
     * Test for GetXMLEncoded
```

```
     *
     * @return void
     */
    public function testGetXMLEncoded ()
    {
        $body = self::getCertificationResponseMessage();

        $expectedXML =
            SGCI_PKIMessage_Body_CertificationResponseTest::
            getXML(
            $body->getCertReqId(), $body->getStatus(), array(),
                null, 'cp'
        );

        $certRepXML = $body->getXMLEncoded();

        $xml = new DomDocument();
        $certRepXML = $xml->importNode($certRepXML, true);
        $xml->appendChild($certRepXML);

        $validSchema = $xml->schemaValidateSource(
            SGCI_PKIMessage_Body_CertificationResponseTest::
                getCertRepMsgXMLSchema()
        );

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());

        return $body;
    }

    /**
     * Returns a
         SGCI_PKIMessage_Body_CertificationResponseMessage
         object
     *
     * @return SGCI_PKIMessage_Body_CertificationResponseMessage
     */
    public static function getCertificationResponseMessage()
    {
        $pkiStatusInfo = new SGCI_PKIStatusInfo();
        $pkiStatusInfo->setStatus(SGCI_PKIStatus::
            STATUS_ACCEPTED);

        $cp = new
            SGCI_PKIMessage_Body_CertificationResponseMessage()
            ;
        $cp->setCertReqId(5);
        $cp->setStatus($pkiStatusInfo);

        return $cp;
    }
```

```
}
```

Código Fonte C.47: Classe SGCI_PKIMessage_Body_CertificationResponseMessageTest

```
<?php
/**
 * Class that tests the
     SGCI_PKIMessage_InitializationRequestMessage class
 */
class SGCI_PKIMessage_Body_InitializationRequestMessageTest
    extends PHPUnit_Framework_TestCase
{
    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncoded ()
    {
        $body = self::getInitializationRequestMessage();

        $expectedXML =
            SGCI_PKIMessage_Body_CertificationRequestMessagesTest
            ::getXML(
            $body->getCertReqMsgs(), 'ir'
        );
        $certReqXML = $body->getXMLEncoded();

        $xml = new DomDocument();

        $certReqXML = $xml->importNode($certReqXML, true);

        $xml->appendChild($certReqXML);

        $validSchema = $xml->schemaValidateSource(
            SGCI_PKIMessage_Body_CertificationRequestMessagesTest
                ::getCertReqMsgXMLSchema()
        );

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Returns a
         SGCI_PKIMessage_Body_InitializationRequestMessage
         object
     *
     * @return SGCI_PKIMessage_Body_InitializationRequestMessage
     */
```

```
    public static function getInitializationRequestMessage()
    {
        $certTemplate = new SGCI_PKIMessage_CertificateTemplate
            ();
        $certTemplate->setSerialNumber(10);
        $certRequest = new
            SGCI_PKIMessage_CertificationRequestMessage();
        $certRequest->setCertReqId(3);
        $certRequest->setCertTemplate($certTemplate);

        $ir = new
            SGCI_PKIMessage_Body_InitializationRequestMessage()
            ;
        $ir->addCertReqMsg($certRequest);

        return $ir;
    }
}
```

Código Fonte C.48: Classe SGCI_PKIMessage_Body_InitializationRequestMessageTest

```
<?php
/**
 * Class that tests the
     SGCI_PKIMessage_Body_InitializationResponseMessage class
 */
class SGCI_PKIMessage_Body_InitializationResponseMessageTest
    extends PHPUnit_Framework_TestCase
{
    /**
     * Test for GetXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncoded ()
    {
        $body = self::getInitializationResponseMessage();

        $expectedXML =
            SGCI_PKIMessage_Body_CertificationResponseTest::
            getXML(
            $body->getCertReqId(), $body->getStatus(), array(),
                null, 'ip'
        );

        $certRepXML = $body->getXMLEncoded();

        $xml = new DomDocument();
        $certRepXML = $xml->importNode($certRepXML, true);
        $xml->appendChild($certRepXML);
```

```
        $validSchema = $xml->schemaValidateSource(
            SGCI_PKIMessage_Body_CertificationResponseTest::
                getCertRepMsgXMLSchema()
        );

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Returns a
       SGCI_PKIMessage_Body_InitializationResponseMessage
       object
     *
     * @return
       SGCI_PKIMessage_Body_InitializationResponseMessage
     */
    public static function getInitializationResponseMessage()
    {
        $certReqId = 5;
        $pkiStatusInfo = new SGCI_PKIStatusInfo();
        $pkiStatusInfo->setStatus(SGCI_PKIStatus::
            STATUS_ACCEPTED);

        $ip = new
            SGCI_PKIMessage_Body_InitializationResponseMessage
            ();

        $ip->setCertReqId($certReqId);
        $ip->setStatus($pkiStatusInfo);

        return $ip;
    }
}
```

Código Fonte C.49: Classe SGCI_PKIMessage_Body_InitializationResponseMessageTest

```
<?php
/**
 * Class that tests the
     SGCI_PKIMessage_Body_ListCertificatesResponse class.
 */
class SGCI_PKIMessage_Body_ListCertificatesResponseTest
    extends PHPUnit_Framework_TestCase
{

    /**
     * Class being tested.
     *
```

```
     * @var SGCI_PKIMessage_Body_ListCertificatesResponse
     */
    protected $_listCertReqMsg;

    /**
     * Instantiate a
          SGCI_PKIMessage_Body_ListCertificatesResponse object
     *
     * @return void
     */
    public function setUp ()
    {
        $this->_listCertReqMsg = new
            SGCI_PKIMessage_Body_ListCertificatesResponse();
    }

    /**
     * Test for loadXML
     *
     * @return void
     */
    public function testCanLoadXMLWithoutCertificate ()
    {
        $xml = $this->_getDomNode(array());

        $this->_listCertReqMsg->loadXML($xml);

        $certificates = $this->_listCertReqMsg->getCertificates
            ();
        $this->assertEquals(0, count($certificates));
    }

    /**
     * Test for loadXML
     *
     * @return void
     */
    public function testCanLoadXMLWithOneCertificate ()
    {
        $certificate = $this->_getCertificate();

        $xml = $this->_getDomNode(array($certificate));

        $this->_listCertReqMsg->loadXML($xml);

        $certificates = $this->_listCertReqMsg->getCertificates
            ();
        $this->assertEquals(1, count($certificates));
        $this->assertEquals($certificate->getPem(),
            $certificates[0]->getPem());
    }
```

```
    /**
     * Test for loadXML
     *
     * @return void
     */
    public function testCanLoadXMLWithTwoCertificates ()
    {
        $certificateA = $this->_getCertificate();
        $certificateB = $this->_getCertificate();
        $expectedCertificates = array(
                                    $certificateA,
                                    $certificateB,
                                );

        $xml = $this->_getDomNode($expectedCertificates);

        $this->_listCertReqMsg->loadXML($xml);

        $certificates = $this->_listCertReqMsg->getCertificates
            ();
        $this->assertEquals(count($expectedCertificates), count(
            $certificates));
        $this->assertEquals($expectedCertificates[1]->getPem(),
            $certificates[0]->getPem());
        $this->assertEquals($expectedCertificates[1]->getPem(),
            $certificates[1]->getPem());
    }

    /**
     * Construct the XML from the parameters and return a
         DomNode representation of the XML
     *
     * @param array
         $certificates The certificates to be
     *

         added to the message
     *
     * @return DOMNode
     */
    protected function _getDomNode (array $certificates = array
        ())
    {
        $xml = $this->_getXML($certificates);

        $dom = new DOMDocument();
        $dom->loadXML($xml);

        return $dom->getElementsByTagName('listCertRep')->item
            (0);
    }
```

```
    /**
     * Test for addCertificate
     *
     * @return void
     */
    public function testCanAddCertificate ()
    {
        $expectedCertificate = $this->_getCertificate();

        $this->_listCertReqMsg->addCertificate(
            $expectedCertificate);

        $certificates = $this->_listCertReqMsg->getCertificates
            ();

        $this->assertEquals(1, count($certificates));
        $this->assertEquals($expectedCertificate->getPem(),
            $certificates[0]->getPem());
    }

    /**
     * Test for addCertificate
     *
     * @return void
     */
    public function testCanAddTwoCertificates ()
    {
        $certificateA = $this->_getCertificate();
        $certificateB = $this->_getCertificate();
        $expectedCertificates = array(
                                  $certificateA,
                                  $certificateB,
                                );

        $this->_listCertReqMsg->addCertificate($certificateA);
        $this->_listCertReqMsg->addCertificate($certificateB);

        $certificates = $this->_listCertReqMsg->getCertificates
            ();

        $this->assertEquals(count($expectedCertificates), count(
            $certificates));
        $this->assertEquals($expectedCertificates[0]->getPem(),
            $certificates[0]->getPem());
        $this->assertEquals($expectedCertificates[1]->getPem(),
            $certificates[1]->getPem());
    }

    /**
     * Test for addCertificates
     *
     * @return void
```

```
     */
    public function testCanAddCertificates ()
    {
        $certificateA = $this->_getCertificate();
        $certificateB = $this->_getCertificate();
        $expectedCertificates = array(
                                    $certificateA,
                                    $certificateB,
                                );

        $this->_listCertReqMsg->addCertificates(
            $expectedCertificates);

        $certificates = $this->_listCertReqMsg->getCertificates
            ();

        $this->assertEquals(count($expectedCertificates), count(
            $certificates));
        $this->assertEquals($expectedCertificates[0]->getPem(),
            $certificates[0]->getPem());
        $this->assertEquals($expectedCertificates[1]->getPem(),
            $certificates[1]->getPem());
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testCanGetXmlEncodedWithoutCertificates ()
    {
        $expectedXml = $this->_getXML();

        $listCertReq = $this->_listCertReqMsg->getXMLEncoded();

        $xml = new DomDocument();

        $listCertReq = $xml->importNode($listCertReq, true);
        $xml->appendChild($listCertReq);
        $validSchema = $xml->schemaValidateSource($this->
            _getGenRepContentXMLSchema());
        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXml, $xml->saveXML());
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testCanGetXmlEncodedWithOneCertificate ()
    {
```

```
        $certificate = $this->_getCertificate();
        $expectedXml = $this->_getXML(array($certificate));

        $this->_listCertReqMsg->addCertificate($certificate);
        $listCertReq = $this->_listCertReqMsg->getXMLEncoded();

        $xml = new DomDocument();

        $listCertReq = $xml->importNode($listCertReq, true);
        $xml->appendChild($listCertReq);
        $validSchema = $xml->schemaValidateSource($this->
            _getGenRepContentXMLSchema());
        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXml, $xml->saveXML());
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testCanGetXmlEncodedWithTwoCertificates()
    {
        $certificateA = $this->_getCertificate();
        $certificateB = $this->_getCertificate();
        $certificates = array(
                            $certificateA,
                            $certificateB,
                        );

        $expectedXml = $this->_getXML($certificates);

        $this->_listCertReqMsg->addCertificates($certificates);
        $listCertReq = $this->_listCertReqMsg->getXMLEncoded();

        $xml = new DomDocument();

        $listCertReq = $xml->importNode($listCertReq, true);
        $xml->appendChild($listCertReq);
        $validSchema = $xml->schemaValidateSource($this->
            _getGenRepContentXMLSchema());
        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXml, $xml->saveXML());
    }

    /**
     * Returns a certificate object
     *
     * @return LabsecCL_Security_Certification_CertificateCL
     */
    protected function _getCertificate()
    {
```

```
        $data = '-----BEGIN CERTIFICATE-----' . PHP_EOL .
            '
                MIIB5TCCAU6gAwIBAgIBATANBgkqhkiG9w0BAQUFADAPMQ0wCwYDVQQDEwRUZXN
                ' . PHP_EOL
            . '
                MB4XDTEwMDgwNjE1MzAwMloXDTIwMDgwMzE1MzAwMlowDzENMAsGA1UEAxMEVGV
                ' . PHP_EOL
            . '
                dDCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAnp1d1wAcIhDFD00IqQyqFmk
                ' . PHP_EOL
            . 'j+
                Jqtyf8aY19Xr8UYYFA98AyKrbQ9rSWu7Cun3C09yRLu4zbLDfALTUxqGVVSQB7
                ' . PHP_EOL
            . 'baOK5HCpo63vNT+
                Ktkrd9Zt1cf86XWgVZI2iOwHX4jacx7QLALo2zdVmmpPDMydf
                ' . PHP_EOL
            . 'h/bg/Kwybe08K/EZO60CAwEAAaNRME8wDwYDVR0TAQH/
                BAUwAwEB/zAOBgNVHQ8B' . PHP_EOL
            . 'Af8EBAMCAQYwHQYDVR00BBYEFJ8KY2/9nEEfNUep/bmBfHG9/
                iRjMA0GA1UdEQQG' . PHP_EOL
            . '
                MASkAjAAMA0GCSqGSIb3DQEBBQUAA4GBAH4LsmdDogNQu5Wej6rQGL8TGgq4kpH
                ' . PHP_EOL
            . 'TugTGQo/94GpS+6vna1bkNIotTmSdNLBPHTOuvBD+
                nL9Ku3tezjmgsPV+BmblEFE' . PHP_EOL
            . 'v58L1GnapdQXfJe+
                fNcpmYZ0TmXwE12TvvL6aWFMfnGniMcbBb4QQg1c10Y2dmjN
                ' . PHP_EOL
            . 'yHP8BB85fMxn' . PHP_EOL
            . '-----END CERTIFICATE-----';

        return new LabsecCL_Security_Certification_CertificateCL
            ($data);
    }

    /**
     * Construct the XML from the parameters and return a string
         representation of the XML
     *
     * @param array  $certificates The certificates to be added
         to the message
     * @param string $nodeName     The name of the root node of
         the XML
     *
     * @return DOMNode
     */
    protected function _getXML (array $certificates = array(),
        $nodeName = 'listCertRep')
    {

        if (empty($certificates) === false) {
            $value = '';
```

```
            foreach ($certificates as $certificate) {
                $value .= '<certificate>' . $certificate->getPem
                    () . '</certificate>';
            }
            $xml = '<?xml version="1.0"?>' . PHP_EOL
                 . '<' . $nodeName . '>'
                 . $value
                 . '</' . $nodeName . '>' . PHP_EOL;
        } else {
            $xml = '<?xml version="1.0"?>' . PHP_EOL
                 . '<' . $nodeName . '/>' . PHP_EOL;
        }

        return $xml;
    }

    /**
     * Returns the schema validator of the genp structure
     *
     * @return string
     */
    protected function _getGenRepContentXMLSchema ()
    {
        return '<?xml version="1.0"?>
                        <xs:schema xmlns:xs="http://www.w3.org
                            /2001/XMLSchema">

                        <xs:element name="listCertRep">
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element name
                                                    ="
                                                    certificate
                                                    " type="xs:
                                                    string"
                                                    minOccurs
                                                    ="0"
                                                    maxOccurs="
                                                    unbounded
                                                    "/>
                                        </xs:sequence>
                                </xs:complexType>
                        </xs:element>

                        </xs:schema>';
    }
}
```

Código Fonte C.50: Classe SGCI_PKIMessage_Body_ListCertificatesResponseTest

```
<?php
/**
 * Class that tests the
     SGCI_PKIMessage_Body_PollingRequestContent class
 */
class SGCI_PKIMessage_Body_PollingRequestContentTest extends
    PHPUnit_Framework_TestCase
{
    /**
     * Class being tested.
     *
     * @var SGCI_PKIMessage_Body_PollingRequestContent
     */
    protected $_pollReqContent;

    /**
     * Instantiate a SGCI_PKIMessage_Body_PollingRequestContent
         object.
     *
     * @return void
     */
    public function setUp ()
    {
        $this->_pollReqContent = new
            SGCI_PKIMessage_Body_PollingRequestContent();
    }

    /**
     * Test for loadXML
     *
     * @return void
     */
    public function testLoadXMLWithAllFields ()
    {
        $expectedCertReqIds = array(
                                15,
                                27,
                              );

        $xml = $this->_getDomNode($expectedCertReqIds);

        $this->_pollReqContent->loadXML($xml);

        $certReqIds = $this->_pollReqContent->getCertReqids();

        $this->assertEquals(count($expectedCertReqIds), count(
            $certReqIds));
        $this->assertEquals($expectedCertReqIds[0], $certReqIds
            [0]);
        $this->assertEquals($expectedCertReqIds[1], $certReqIds
            [1]);
    }
```

```
/**
 * Construct the XML from the parameters and return a
     DomNode representation of the XML
 *
 * @param array $certReqIds The ids to poll
 *
 * @return void
 */
protected function _getDomNode (array $certReqIds)
{
    $xml = self::getXML($certReqIds);

    $dom = new DOMDocument();
    $dom->loadXML($xml);

    return $dom->getElementsByTagName('PollReqContent')->
        item(0);
}

/**
 * Test for addCertReqId
 *
 * @param integer $id The id provided by the dataProvider
 *
 * @dataProvider invalidIdProvider
 * @return void
 */
public function testAddCertReqIdWithInvalidId ($id)
{
    $this->setExpectedException('
        SGCI_Exception_InvalidParameter');

    $this->_pollReqContent->addCertReqId($id);
}

/**
 * Data Provider with invalid Ids
 *
 * @return mixed
 */
public function invalidIdProvider ()
{
    $array = array(
              array('invalid'),
              array(new SGCI_PKIMessage_CertificateTemplate
                  ()),
              array(false),
             );

    return $array;
}
```

```
    /**
     * Test for addCertReqId
     *
     * @return void
     */
    public function testCanAddOneCertReqId ()
    {
        $expectedId = array(2);

        $this->_pollReqContent->addCertReqId($expectedId[0]);
        $certReqIds = $this->_pollReqContent->getCertReqids();

        $this->assertEquals(count($expectedId), count(
            $certReqIds));
        $this->assertEquals($expectedId[0], $certReqIds[0]);
    }

    /**
     * Test for addCertReqId
     *
     * @return void
     */
    public function testCanAddTwoCertReqids ()
    {
        $expectedId = array(
                        2,
                        10,
                      );

        $this->_pollReqContent->addCertReqId($expectedId[0]);
        $this->_pollReqContent->addCertReqId($expectedId[1]);
        $certReqIds = $this->_pollReqContent->getCertReqids();

        $this->assertEquals(count($expectedId), count(
            $certReqIds));
        $this->assertEquals($expectedId[0], $certReqIds[0]);
        $this->assertEquals($expectedId[1], $certReqIds[1]);
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithoutCertReqIds ()
    {
        $this->setExpectedException('SGCI_Exception');

        $this->_pollReqContent->getXMLEncoded();
    }
```

```
/**
 * Test for getXMLEncoded
 *
 * @return void
 */
public function testGetXMLEncoded ()
{
    $certReqIds = array(
                   5,
                   35,
                  );

    foreach ($certReqIds as $certReqId) {
        $this->_pollReqContent->addCertReqId($certReqId);
    }

    $expectedXML = self::getXML($certReqIds);

    $pollReqXML = $this->_pollReqContent->getXMLEncoded();

    $xml = new DomDocument();
    $pollReqXML = $xml->importNode($pollReqXML, true);

    $xml->appendChild($pollReqXML);

    $validSchema = $xml->schemaValidateSource(self::
        getPollReqContentXMLSchema());

    $this->assertTrue($validSchema);
    $this->assertEquals($expectedXML, $xml->saveXML());
}

/**
 * Construct the XML from the parameters and return a string
      representation of the XML
 *
 * @param array $certReqIds The ids to poll
 * @param string $nodeName  The name of the root node of the
      generated XML
 *
 * @return string
 */
public static function getXML(array $certReqIds, $nodeName =
     'PollReqContent')
{
    $certReqIdsXML = '';
    foreach ($certReqIds as $certReqId) {
        $certReqIdsXML .= '<certReqId>' . $certReqId . '</
            certReqId>';
    }

    $xml = '<?xml version="1.0"?>' . PHP_EOL
```

```
            . '<' . $nodeName . '>'
            . $certReqIdsXML
            . '</' . $nodeName . '>' . PHP_EOL;

        return $xml;
    }

    /**
     * Returns the XML schema for the RevReqContent structure
     *
     * @return string
     */
    public static function getPollReqContentXMLSchema ()
    {
            return '<?xml version="1.0"?>
                    <xs:schema xmlns:xs="http://www.w3.org
                        /2001/XMLSchema">

                    <xs:element name="pollReq"
                        substitutionGroup="PollReqContent
                        "/>

                    <xs:element name="PollReqContent">
                            <xs:complexType>
                                    <xs:sequence>
                                            <xs:element name
                                                ="certReqId
                                                " maxOccurs
                                                ="unbounded
                                                " type="xs:
                                                integer"/>
                                    </xs:sequence>
                            </xs:complexType>
                    </xs:element>

                    </xs:schema>';
    }
}
```

Código Fonte C.51: Classe SGCI_PKIMessage_Body_PollingRequestContentTest

```
<?php
/**
 * Class that tests the
     SGCI_PKIMessage_Body_PollingRequestMessage class
 */
class SGCI_PKIMessage_Body_PollingRequestMessageTest extends
    PHPUnit_Framework_TestCase
{
    /**
```

```
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncoded ()
    {
        $certReqIds = array(
                        15,
                        20,
                      );

        $pollReq = new
            SGCI_PKIMessage_Body_PollingRequestMessage();
        foreach ($certReqIds as $certReqId) {
            $pollReq->addCertReqId($certReqId);
        }

        $expectedXML =
            SGCI_PKIMessage_Body_PollingRequestContentTest::
            getXML(
            $certReqIds, 'pollReq'
        );

        $pollReqXML = $pollReq->getXMLEncoded();

        $xml = new DomDocument();
        $pollReqXML = $xml->importNode($pollReqXML, true);

        $xml->appendChild($pollReqXML);

        $validSchema = $xml->schemaValidateSource(
            SGCI_PKIMessage_Body_PollingRequestContentTest::
                getPollReqContentXMLSchema()
        );

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }
}
```

Código Fonte C.52: Classe SGCI_PKIMessage_Body_PollingRequestMessageTest

```
<?php
/**
 * Class that tests the
     SGCI_PKIMessage_Body_PollingResponseContent class
 */
class SGCI_PKIMessage_Body_PollingResponseContentTest extends
    PHPUnit_Framework_TestCase
{
```

```
    /**
     * Class being tested.
     *
     * @var SGCI_PKIMessage_Body_PollingResponseContent
     */
    protected $_pollRepContent;

    /**
     * Instantiate a SGCI_PKIMessage_CertificationRequestMessage
          object.
     *
     * @return void
     */
    public function setUp ()
    {
        $this->_pollRepContent = new
            SGCI_PKIMessage_Body_PollingResponseContent();
    }

    /**
     * Test for loadXML
     *
     * @return void
     */
    public function testLoadXMLWithOnlyRequiredFields ()
    {
        $expectedCertReqId = 15;
        $expectedCheckAfter = 100;

        $xml = $this->_getDomNode($expectedCertReqId,
            $expectedCheckAfter);

        $this->_pollRepContent->loadXML($xml);

        $response = $this->_pollRepContent->getPollingResponses
            ();

        $this->assertEquals(1, count($response));
        $this->assertEquals($expectedCertReqId, $response[0]['
            certReqId']);
        $this->assertEquals($expectedCheckAfter, $response[0]['
            checkAfter']);
        $this->assertEquals(null, $response[0]['reason']);
    }

    /**
     * Test for loadXML
     *
     * @return void
     */
    public function testLoadXMLWithAllFields ()
    {
```

```
        $expectedCertReqId = 15;
        $expectedCheckAfter = 100;
        $expectedReason = 'Try later';

        $xml = $this->_getDomNode($expectedCertReqId,
            $expectedCheckAfter, $expectedReason);

        $this->_pollRepContent->loadXML($xml);

        $response = $this->_pollRepContent->getPollingResponses
            ();

        $this->assertEquals(1, count($response));
        $this->assertEquals($expectedCertReqId, $response[0]['
            certReqId']);
        $this->assertEquals($expectedReason, $response[0]['
            reason']);
        $this->assertEquals($expectedCheckAfter, $response[0]['
            checkAfter']);
    }

    /**
     * Construct the XML from the parameters and return a
         DomNode representation of the XML
     *
     * @param integer $certReqId  The id of the request
     * @param integer $checkAfter Time, in seconds, for the
         procuction of the next message
     * @param string  $reason     A human readable about the
         production of this message
     *
     * @return void
     */
    protected function _getDomNode ($certReqId, $checkAfter,
        $reason = null)
    {
        $xml = self::getXML($certReqId, $checkAfter, $reason);

        $dom = new DOMDocument();
        $dom->loadXML($xml);

        return $dom->getElementsByTagName('PollRepContent')->
            item(0);
    }

    /**
     * Test for addPollingResponse
     *
     * @param integer $certReqId The id of the request
     *
     * @dataProvider invalidIntegerProvider
     * @return void
```

```
     */
    public function testAddPollingResponseWithInvalidCertReqId (
        $certReqId)
    {
        $this->setExpectedException('
            SGCI_Exception_InvalidParameter');

        $this->_pollRepContent->addPollingResponse($certReqId,
            1);
    }

    /**
     * Test for addPollingResponse
     *
     * @param integer $checkAfter Time, in seconds, for the
         procuction of the next message
     *
     * @dataProvider invalidIntegerProvider
     * @return void
     */
    public function testAddPollingResponseWithInvalidCheckAfter
        ($checkAfter)
    {
        $this->setExpectedException('
            SGCI_Exception_InvalidParameter');

        $this->_pollRepContent->addPollingResponse(1,
            $checkAfter);
    }

    /**
     * Data Provider with invalid integers
     *
     * @return mixed
     */
    public function invalidIntegerProvider ()
    {
        $array = array(
                  array('invalid'),
                  array(new SGCI_PKIMessage_CertificateTemplate
                      ()),
                  array(false),
                 );

        return $array;
    }

    /**
     * Test for addPollingResponse
     *
     * @param string $reason Invalid reasons values provided by
         the dataProvider
```

```
     *
     * @dataProvider invalidStringProvider
     * @return void
     */
    public function testAddpollingResponseWithInvalidReason (
        $reason)
    {
        $this->setExpectedException('
            SGCI_Exception_InvalidParameter');

        $this->_pollRepContent->addPollingResponse(1, 1, $reason
            );
    }

    /**
     * Data Provider with invalid strings
     *
     * @return mixed
     */
    public function invalidStringProvider ()
    {
        $array = array(
                  array(1),
                  array(new SGCI_PKIMessage_CertificateTemplate
                      ()),
                  array(false),
                 );

        return $array;
    }

    /**
     * Test for addPollingResponse
     *
     * @return void
     */
    public function testAddResponseWithoutReason ()
    {
        $expectedCertReqId = 12;
        $expectedCheckAfter = 150;

        $this->_pollRepContent->addPollingResponse(
            $expectedCertReqId, $expectedCheckAfter);
        $response = $this->_pollRepContent->getPollingResponses
            ();

        $this->assertEquals(1, count($response));
        $this->assertEquals($expectedCertReqId, $response[0]['
            certReqId']);
        $this->assertEquals($expectedCheckAfter, $response[0]['
            checkAfter']);
        $this->assertEquals(null, $response[0]['reason']);
```

```
    }

    /**
     * Test for addPollingResponse
     *
     * @return void
     */
    public function testAddResponseWithAllFieds ()
    {
        $expectedCertReqId = 12;
        $expectedCheckAfter = 150;
        $expectedReason = 'Please try later';

        $this->_pollRepContent->addPollingResponse(
            $expectedCertReqId, $expectedCheckAfter,
                $expectedReason
        );

        $response = $this->_pollRepContent->getPollingResponses
            ();

        $this->assertEquals(1, count($response));
        $this->assertEquals($expectedCertReqId, $response[0]['
            certReqId']);
        $this->assertEquals($expectedCheckAfter, $response[0]['
            checkAfter']);
        $this->assertEquals($expectedReason, $response[0]['
            reason']);
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithoutResponse ()
    {
        $this->setExpectedException('SGCI_Exception');

        $this->_pollRepContent->getXMLEncoded();
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithoutReason ()
    {
        $certReqId = 15;
        $checkAfter = 100;
```

```
        $this->_pollRepContent->addPollingResponse($certReqId,
            $checkAfter);

        $expectedXML = self::getXML($certReqId, $checkAfter);

        $pollRepXML = $this->_pollRepContent->getXMLEncoded();

        $xml = new DomDocument();
        $pollRepXML = $xml->importNode($pollRepXML, true);

        $xml->appendChild($pollRepXML);

        $validSchema = $xml->schemaValidateSource(self::
            getPollRepContentXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithallFields ()
    {
        $certReqId = 15;
        $checkAfter = 100;
        $reason = 'This is a reason';

        $this->_pollRepContent->addPollingResponse($certReqId,
            $checkAfter, $reason);

        $expectedXML = self::getXML($certReqId, $checkAfter,
            $reason);

        $pollRepXML = $this->_pollRepContent->getXMLEncoded();

        $xml = new DomDocument();
        $pollRepXML = $xml->importNode($pollRepXML, true);

        $xml->appendChild($pollRepXML);

        $validSchema = $xml->schemaValidateSource(self::
            getPollRepContentXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Construct the XML from the parameters and return a string
```

```
        representation of the XML
     *
     * @param integer $certReqId  The id of the request
     * @param integer $checkAfter Time, in seconds, for the
        procuction of the next message
     * @param string  $reason     A human readable about the
        production of this message
     * @param string $nodeName    The name of the root node of
        the generated XML
     *
     * @return string
     */
    public static function getXML
    (
        $certReqId,
        $checkAfter,
        $reason = null,
        $nodeName = 'PollRepContent'
    ) {
        $reasonXML = '';
        if ($reason !== null) {
            $reasonXML = '<reason>' . $reason . '</reason>';
        }
        $xml = '<?xml version="1.0"?>' . PHP_EOL
             . '<' . $nodeName . '>'
             . '<sequence>'
             . '<certReqId>' . $certReqId . '</certReqId>'
             . '<checkAfter>' . $checkAfter . '</checkAfter>'
             . $reasonXML
             . '</sequence>'
             . '</' . $nodeName . '>' . PHP_EOL;

        return $xml;
    }

    /**
     * Returns the XML schema for the RevReqContent structure
     *
     * @return string
     */
    public static function getPollRepContentXMLSchema ()
    {
                return '<?xml version="1.0"?>
                        <xs:schema xmlns:xs="http://www.w3.org
                            /2001/XMLSchema">

                        <xs:element name="pollRep"
                            substitutionGroup="PollRepContent
                            "/>

                        <xs:element name="PollRepContent">
                                <xs:complexType>
```

```
                                        <xs:sequence>
                                                <xs:element name
                                                    ="sequence"
                                                     maxOccurs
                                                    ="unbounded
                                                    "/>
                                        </xs:sequence>
                                </xs:complexType>
                        </xs:element>

                        <xs:element name="sequence">
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element name
                                                    ="certReqId
                                                    " type="xs:
                                                    integer"/>
                                                <xs:element name
                                                    ="
                                                    checkAfter"
                                                     type="xs:
                                                    integer"/>
                                                <xs:element name
                                                    ="reason"
                                                    minOccurs
                                                    ="0" type="
                                                    xs:string
                                                    "/>
                                        </xs:sequence>
                                </xs:complexType>
                        </xs:element>

                        </xs:schema>';
    }
}
```

Código Fonte C.53: Classe SGCI_PKIMessage_Body_PollingResponseContentTest

```
<?php
/**
 * Class that tests the
     SGCI_PKIMessage_Body_PollingResponseMessage class
 */
class SGCI_PKIMessage_Body_PollingResponseMessageTest extends
    PHPUnit_Framework_TestCase
{
    /**
     * Test for getXMLEncoded
     *
     * @return void
```

```
     */
    public function testGetXMLEncoded ()
    {
        $certReqId = 125;
        $checkAfter = 500;
        $reason = 'this is a reason';

        $pollRepContent = new
            SGCI_PKIMessage_Body_PollingResponseMessage();
        $pollRepContent->addPollingResponse($certReqId,
            $checkAfter, $reason);

        $expectedXML =
            SGCI_PKIMessage_Body_PollingResponseContentTest::
            getXML(
            $certReqId, $checkAfter, $reason, 'pollRep'
        );

        $pollRepXML = $pollRepContent->getXMLEncoded();

        $xml = new DomDocument();
        $pollRepXML = $xml->importNode($pollRepXML, true);

        $xml->appendChild($pollRepXML);

        $validSchema = $xml->schemaValidateSource(
            SGCI_PKIMessage_Body_PollingResponseContentTest::
                getPollRepContentXMLSchema()
        );

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }
}
```

Código Fonte C.54: Classe SGCI_PKIMessage_Body_PollingResponseMessageTest

```
<?php
/**
 * Class that tests the
     SGCI_PKIMessage_Body_RevocationRequestContent class
 */
class SGCI_PKIMessage_Body_RevocationRequestContentTest extends
    PHPUnit_Framework_TestCase
{
    /**
     * Class being tested.
     *
     * @var SGCI_PKIMessage_Body_RevocationRequestMessage
     */
```

```
protected $_revRequest;

/**
 * Instantiate a
     SGCI_PKIMessage_Body_RevocationRequestContent object.
 *
 * @return void
 */
public function setUp ()
{
    $this->_revRequest = new
        SGCI_PKIMessage_Body_RevocationRequestContent();
}

/**
 * Test for loadXML
 *
 * @return void
 */
public function testLoadXMLWithOnlyRequiredFields ()
{
    $certTemplate = new SGCI_PKIMessage_CertificateTemplate
        ();
    $certTemplate->setSerialNumber(1);
    $xml = $this->_getDomNode($certTemplate);

    $this->_revRequest->loadXML($xml);

    $revReqs = $this->_revRequest->getRevocationRequests();
    $revReq = $revReqs[0];

    $this->assertEquals(1, count($revReqs));
    $this->assertEquals(1, count($revReq));

    $key = SGCI_PKIMessage_Body_RevocationRequestContent::
        CERT_TEMPLATE;
    $this->assertEquals($certTemplate, $revReq[$key]);
}

/**
 * Test for loadXML
 *
 * @return void
 */
public function testLoadXMLWithAllFields ()
{
    $certTemplate = new SGCI_PKIMessage_CertificateTemplate
        ();
    $certTemplate->setSerialNumber(1);
    $crlEntryExtension =
        new
            LabsecCL_Security_Certification_Extension_CRLReasonCodeEx
```

```
                ();

        $code =
            LabsecCL_Security_Certification_Extension_CRLReasonCodeExtensionCL
            ::UNSPECIFIED;
        $crlEntryExtension->setCode($code);
        $crlEntryExtension->setCritical(false);

        $xml = $this->_getDomNode($certTemplate, array(
            $crlEntryExtension));

        $this->_revRequest->loadXML($xml);

        $revReqs = $this->_revRequest->getRevocationRequests();

        $key = SGCI_PKIMessage_Body_RevocationRequestContent::
            CERT_TEMPLATE;
        $actualCertTemplate = $revReqs[0][$key];

        $key = SGCI_PKIMessage_Body_RevocationRequestContent::
            CRL_ENTRY_DETAILS;
        $actualExtensions = $revReqs[0][$key];

        $this->assertEquals(1, count($revReqs));
        $this->assertEquals($certTemplate, $actualCertTemplate);
        $this->assertEquals($crlEntryExtension,
            $actualExtensions[0]);
    }

    /**
     * Construct the XML from the parameters and return a
         DomNode representation of the XML
     *
     * @param SGCI_PKIMessage_CertificateTemplate $certDetails
             Certificate information
     * @param array
         $crlEntryDetails Extensions for the CRL Entry
     *
     * @return void
     */
    protected function _getDomNode
    (
        SGCI_PKIMessage_CertificateTemplate $certDetails,
        array $crlEntryDetails = array()
    ) {
        $xml = self::getXML($certDetails, $crlEntryDetails);
        $dom = new DOMDocument();
        $dom->loadXML($xml);
        return $dom->getElementsByTagName('RevReqContent')->item
            (0);
    }
```

```
/**
 * Test for AddRevocationRequest
 *
 * @return void
 */
public function
    testCanAddRevocationRequestWithoutCrlEntryDetail ()
{
    $certTemplate = new SGCI_PKIMessage_CertificateTemplate
        ();
    $certTemplate->setSerialNumber(1);
    $this->_revRequest->addRevocationRequest($certTemplate);

    $revRequests = $this->_revRequest->getRevocationRequests
        ();

    $this->assertEquals(1, count($revRequests));
    $revRequest = $revRequests[0];

    $key = SGCI_PKIMessage_Body_RevocationRequestContent::
        CERT_TEMPLATE;
    $this->assertEquals($certTemplate, $revRequest[$key]);

    $key = SGCI_PKIMessage_Body_RevocationRequestContent::
        CRL_ENTRY_DETAILS;
    $this->assertArrayNotHasKey($key, $revRequest);
}

/**
 * Test for AddRevocationRequest
 *
 * @return void
 */
public function
    testAddRevocationRequestWithEmptyCertTemplateShouldThrowException
     ()
{
    $this->setExpectedException('
        SGCI_Exception_InvalidParameter');

    $certTemplate = new SGCI_PKIMessage_CertificateTemplate
        ();
    $this->_revRequest->addRevocationRequest($certTemplate);
}

/**
 * Test for AddRevocationRequest
 *
 * @param mixed $crlEntryDetails Invalid values provided by
     the dataProvider
 *
 * @dataProvider invalidcrlEntryDetailsProvider
```

```
     * @return void
     */
    public function
        testAddRevocationRequestWithInvalidCrlEntryDetailShouldThrowException

    (
        $crlEntryDetails
    ) {
        $this->setExpectedException('
            SGCI_Exception_InvalidParameter');

        $certTemplate = new SGCI_PKIMessage_CertificateTemplate
            ();
        $certTemplate->setSerialNumber(1);
        $this->_revRequest->addRevocationRequest($certTemplate,
            $crlEntryDetails);
    }

    /**
     * DataProvider with invalid crlEntryDetail
     *
     * @return mixed
     */
    public function invalidcrlEntryDetailsProvider ()
    {
        return array(
                array(array('invalid')),
                array(
                 array(
                  'invalid',
                  new SGCI_PKIMessage_CertificateTemplate(),
                 ),
                ),
                array(array(new
                    SGCI_PKIMessage_CertificateTemplate())),
               );
    }

    /**
     * Test for AddRevocationRequest
     *
     * @return void
     */
    public function testCanAddRevocationRequest ()
    {
        $certTemplate = new SGCI_PKIMessage_CertificateTemplate
            ();
        $certTemplate->setSerialNumber(1);

        $crlEntryExtension =
            new
                LabsecCL_Security_Certification_Extension_CRLReasonCodeExtensio
```

```
            ();

        $code =
            LabsecCL_Security_Certification_Extension_CRLReasonCodeExtens
            ::KEY_COMPROMISE;
        $crlEntryExtension->setCode($code);
        $this->_revRequest->addRevocationRequest($certTemplate,
            array($crlEntryExtension));

        $revRequests = $this->_revRequest->getRevocationRequests
            ();

        $this->assertEquals(1, count($revRequests));
        $revRequest = $revRequests[0];

        $key = SGCI_PKIMessage_Body_RevocationRequestContent::
            CERT_TEMPLATE;
        $this->assertEquals($certTemplate, $revRequest[$key]);

        $key = SGCI_PKIMessage_Body_RevocationRequestContent::
            CRL_ENTRY_DETAILS;
        $this->assertEquals($crlEntryExtension, $revRequest[$key
            ][0]);
    }

    /**
     * Test for AddRevocationRequest
     *
     * @return void
     */
    public function testCanAddTwoRevocationRequests()
    {
        $certTemplateOne = new
            SGCI_PKIMessage_CertificateTemplate();
        $certTemplateOne->setSerialNumber(1);
        $crlEntryExtensionOne =
            new
                LabsecCL_Security_Certification_Extension_CRLReasonCodeEx
                ();

        $code =
            LabsecCL_Security_Certification_Extension_CRLReasonCodeExtens
            ::KEY_COMPROMISE;
        $crlEntryExtensionOne->setCode($code);

        $this->_revRequest->addRevocationRequest(
            $certTemplateOne, array($crlEntryExtensionOne));

        $certTemplateTwo = new
            SGCI_PKIMessage_CertificateTemplate();
        $certTemplateTwo->setSerialNumber(20);
```

```
        $crlEntryExtensionTwo =
            new
                LabsecCL_Security_Certification_Extension_CRLReasonCodeExtensio
                ();

        $code =
            LabsecCL_Security_Certification_Extension_CRLReasonCodeExtensionCL
            ::KEY_COMPROMISE;
        $crlEntryExtensionTwo->setCode($code);

        $crlEntryExtensionThree =
            new
                LabsecCL_Security_Certification_Extension_CRLReasonCodeExtensio
                ();

        $code =
            LabsecCL_Security_Certification_Extension_CRLReasonCodeExtensionCL
            ::UNSPECIFIED;
        $crlEntryExtensionThree->setCode($code);

        $array = array(
                  $crlEntryExtensionTwo,
                  $crlEntryExtensionThree,
                 );
        $this->_revRequest->addRevocationRequest(
            $certTemplateTwo, $array);

        $revRequests = $this->_revRequest->getRevocationRequests
            ();

        $this->assertEquals(2, count($revRequests));
        $revRequest = $revRequests[0];

        $certTemplateKey =
            SGCI_PKIMessage_Body_RevocationRequestContent::
            CERT_TEMPLATE;
        $crlEntryKey =
            SGCI_PKIMessage_Body_RevocationRequestContent::
            CRL_ENTRY_DETAILS;
        $this->assertEquals($certTemplateOne, $revRequest[
            $certTemplateKey]);

        $this->assertEquals($crlEntryExtensionOne, $revRequest[
            $crlEntryKey][0]);
        $revRequest = $revRequests[1];

        $this->assertEquals($certTemplateTwo, $revRequest[
            $certTemplateKey]);

        $this->assertEquals(2, count($revRequest[$crlEntryKey]))
            ;
```

```
        $this->assertEquals($crlEntryExtensionTwo, $revRequest[
            $crlEntryKey][0]);
        $this->assertEquals($crlEntryExtensionThree, $revRequest
            [$crlEntryKey][1]);
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function
        testGetXMLEncodedWithoutCertDetailsShouldThrowException
          ()
    {
        $this->setExpectedException('SGCI_Exception');

        $this->_revRequest->getXMLEncoded();
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithOnlyRequiredFields ()
    {
        $certTemplate = new SGCI_PKIMessage_CertificateTemplate
            ();
        $certTemplate->setSerialNumber(1);
        $expectedXML = self::getXML($certTemplate);

        $this->_revRequest->addRevocationRequest($certTemplate);

        $revReqXML = $this->_revRequest->getXMLEncoded();

        $xml = new DomDocument();
        $revReqXML = $xml->importNode($revReqXML, true);
        $xml->appendChild($revReqXML);

        $validSchema = $xml->schemaValidateSource(self::
            getRevReqMsgXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
```

```
    public function testGetXMLEncoded ()
    {
        $certTemplate = new SGCI_PKIMessage_CertificateTemplate
            ();
        $certTemplate->setSerialNumber(5);

        $crlEntryExtension =
            new
                LabsecCL_Security_Certification_Extension_CRLReasonCodeExtensio
                ();
        $crlEntryExtension->setCode(
            LabsecCL_Security_Certification_Extension_CRLReasonCodeExtensionCL
                ::UNSPECIFIED
        );
        $crlEntryExtension->setCritical(false);

        $expectedXML = self::getXML($certTemplate, array(
            $crlEntryExtension));

        $this->_revRequest->addRevocationRequest($certTemplate,
            array($crlEntryExtension));

        $revReqXML = $this->_revRequest->getXMLEncoded();

        $xml = new DomDocument();

        $revReqXML = $xml->importNode($revReqXML, true);
        $xml->appendChild($revReqXML);

        $validSchema = $xml->schemaValidateSource(self::
            getRevReqMsgXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Construct the XML from the parameters and return a string
          representation of the XML
     *
     * @param SGCI_PKIMessage_CertificateTemplate $certDetails
              Certificate information
     * @param array
         $crlEntryDetails Extensions for the CRL Entry
     * @param string                                    $nodeName
                The name of the root node of
     *

         the generated XML
     *
     * @throws Exception
     * @return string
```

```
     */
    public static function getXML
    (
        SGCI_PKIMessage_CertificateTemplate $certDetails,
        array $crlEntryDetails = array(),
        $nodeName = 'RevReqContent'
    ) {
        $certDetailsXML = $certDetails->getXMLEncoded('
            certDetails');

        $doc = new DOMDocument();

        $certDetailsXML = $doc->importNode($certDetailsXML, true
            );
        $certDetailsXML = $doc->saveXML($certDetailsXML);

        $crlEntryDetailsXML = '';

        if (empty($crlEntryDetails) === false) {
            $extensionsXML = '';
            foreach ($crlEntryDetails as $extension) {
                if ($extension instanceof
                    Labsec_Security_Certification_Extension) {
                    $extensionXML = $extension->getXMLEncoded();

                    $doc = new DOMDocument();

                    $extensionXML = $doc->importNode(
                        $extensionXML, true);
                    $extensionXML = $doc->saveXML($extensionXML)
                        ;
                    $extensionsXML .= $extensionXML;
                } else {
                    throw new Exception(
                        'Each crlEntryDetails must be an
                            instance of' . ' '
                        . '
                            Labsec_Security_Certification_Extension
                            '
                    );
                }
            }
            $crlEntryDetailsXML = '<crlEntryDetails>' .
                $extensionsXML . '</crlEntryDetails>';
        }

        $xml = '<?xml version="1.0"?>' . PHP_EOL
             . '<' . $nodeName . '>'
             . '<RevDetails>' . $certDetailsXML .
                 $crlEntryDetailsXML . '</RevDetails>'
             . '</' . $nodeName . '>' . PHP_EOL;
```

```
        return $xml;
    }

    /**
     * Returns the XML schema for the RevReqContent structure
     *
     * @return string
     */
    public static function getRevReqMsgXMLSchema ()
    {
        return '<?xml version="1.0"?>
                        <xs:schema xmlns:xs="http://www.w3.org
                            /2001/XMLSchema">

                        <xs:element name="rr" substitutionGroup
                            ="RevReqContent"/>

                        <xs:element name="RevReqContent">
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element ref
                                                    ="
                                                    RevDetails"
                                                     maxOccurs
                                                    ="unbounded
                                                    "/>
                                        </xs:sequence>
                                </xs:complexType>
                        </xs:element>

                        <xs:element name="RevDetails">
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element ref
                                                    ="
                                                    certDetails
                                                    "/>
                                                <xs:element ref
                                                    ="
                                                    crlEntryDetails
                                                    " minOccurs
                                                    ="0"/>
                                        </xs:sequence>
                                </xs:complexType>
                        </xs:element>

                        <xs:element name="certDetails"
                            substitutionGroup="CertTemplate"/>
                        <xs:element name="crlEntryDetails"
                            substitutionGroup="Extensions"/>

                        <xs:element name="Extensions">
```

```
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref
                            ="Extension
                            " maxOccurs
                            ="unbounded
                            "/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="Extension">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name
                            ="extnID"
                            type="xs:
                            string"/>
                        <xs:element name
                            ="critical"
                             type="xs:
                            boolean"/>
                        <xs:element ref
                            ="extnValue
                            "/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="extnValue">
        <xs:complexType>
                <xs:choice>
                        <xs:element ref
                            ="
                            reasonCode
                            "/>
                </xs:choice>
        </xs:complexType>
</xs:element>

<xs:element name="reasonCode"
    substitutionGroup="ReasonCode"/>

<xs:element name="ReasonCode">
        <xs:simpleType>
                <xs:restriction base="xs
                    :integer">
                        <xs:enumeration
                            value="0"/>
                        <xs:enumeration
                            value="1"/>
                        <xs:enumeration
```

```
                                        value="2"/>
                                    <xs:enumeration
                                        value="3"/>
                                    <xs:enumeration
                                        value="4"/>
                                    <xs:enumeration
                                        value="5"/>
                                    <xs:enumeration
                                        value="6"/>
                                    <xs:enumeration
                                        value="8"/>
                                    <xs:enumeration
                                        value="9"/>
                                    <xs:enumeration
                                        value
                                        ="10"/>
                            </xs:restriction>
                        </xs:simpleType>
                </xs:element>

                <xs:element name="CertTemplate">
                        <xs:complexType>
                                <xs:sequence>
                                        <xs:element ref
                                            ="version"
                                            minOccurs
                                            ="0"/> <!--
                                             Optional
                                            -->
                                        <xs:element name
                                            ="
                                            serialNumber
                                            " minOccurs
                                            ="0" type="
                                            xs:integer
                                            "/> <!--
                                            Optional
                                            -->
                                        <xs:element ref
                                            ="subject"
                                            minOccurs
                                            ="0"/> <!--
                                             Optional
                                            -->
                                        <xs:element name
                                            ="publicKey
                                            " minOccurs
                                            ="0" type="
                                            xs:string
                                            "/> <!--
                                            Optional
                                            -->
```

```
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="version"
    substitutionGroup="Version"/>

<xs:element name="Version">
        <xs:simpleType>
                <xs:restriction base="xs
                    :integer">
                        <xs:enumeration
                            value="0"/>
                             <!-- v1
                            -->
                        <xs:enumeration
                            value="1"/>
                             <!-- v2
                            -->
                        <xs:enumeration
                            value="2"/>
                             <!-- v3
                            -->
            </xs:restriction>
        </xs:simpleType>
</xs:element>

<xs:element name="subject"
    substitutionGroup="RDNSequence"/>

<xs:element name="RDNSequence">
        <xs:complexType>
                <xs:sequence>
                        <xs:choice>
                                <xs:
                                    element

                                    name
                                    ="C
                                    "
                                    type
                                    ="
                                    xs:
                                    string
                                    "/>

                                    <!--

                                    Country

                                    -->
                            <xs:
```

```
                                                    element
                                                    name
                                                    ="L
                                                    "
                                                    type
                                                    ="
                                                    xs:
                                                    string
                                                    "/>
                                                    <!--
                                                    Locality
                                                    -->
                                                <xs:
                                                    element
                                                    name
                                                    ="
                                                    ST"
                                                    type
                                                    ="
                                                    xs:
                                                    string
                                                    "/>
                                                    <!--
                                                    State
                                                     or
                                                    province
                                                    -->
                                                <xs:
                                                    element
                                                    name
                                                    ="O
                                                    "
                                                    type
                                                    ="
                                                    xs:
                                                    string
                                                    "/>
                                                    <!--
                                                    Organization
```

```
    -->
<xs:
    element

    name
    ="
    OU"

    type
    ="
    xs:
    string
    "/>

    <!--

    Organization

    Unit

    -->
<xs:
    element

    name
    ="
    CN"

    type
    ="
    xs:
    string
    "/>

    <!--

    Common

    Name

    -->
<xs:
    element

    name
    ="
    STREET
    "
    type
    ="
    xs:
```

```
                                                    string
                                                    "/>

                                                    <!--

                                                    Street

                                                    Address

                                                    -->
                                                <xs:
                                                    element

                                                    name
                                                    ="E
                                                    "
                                                    type
                                                    ="
                                                    xs:
                                                    string
                                                    "/>

                                                    <!--
                                                     E-
                                                    mail

                                                    Address

                                                    -->
                                            </xs:choice>
                                        </xs:sequence>
                                    </xs:complexType>
                            </xs:element>
                            </xs:schema>';
    }
}
```

Código Fonte C.55: Classe SGCI_PKIMessage_Body_RevocationRequestContentTest

```
<?php
/**
 * Class that tests the
     SGCI_PKIMessage_Body_RevocationRequestMessage class
 */
class SGCI_PKIMessage_Body_RevocationRequestMessageTest extends
    PHPUnit_Framework_TestCase
{
    /**
     * Test for getXMLEncoded
     *
```

```
     * @return void
     */
    public function testGetXMLEncoded ()
    {
        $body = self::getRevocationRequestMessage();

        $revDetails = $body->getRevocationRequests();
        $expectedXML =
            SGCI_PKIMessage_Body_RevocationRequestContentTest::
            getXML(
            $revDetails[0]['certTemplate'], $revDetails[0]['
                crlEntryDetails'], 'rr'
        );

        $revReqXML = $body->getXMLEncoded();

        $xml = new DomDocument();

        $revReqXML = $xml->importNode($revReqXML, true);
        $xml->appendChild($revReqXML);

        $validSchema = $xml->schemaValidateSource(
            SGCI_PKIMessage_Body_RevocationRequestContentTest::
                getRevReqMsgXMLSchema()
        );

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Returns a SGCI_PKIMessage_Body_RevocationRequestMessage
          object
     *
     * @return SGCI_PKIMessage_Body_RevocationRequestMessage
     */
    public static function getRevocationRequestMessage()
    {
        $certTemplate = new SGCI_PKIMessage_CertificateTemplate
            ();
        $certTemplate->setSerialNumber(20);

        $crlEntryExtension =
            new
                LabsecCL_Security_Certification_Extension_CRLReasonCodeEx
                ();
        $crlEntryExtension->setCode(
            LabsecCL_Security_Certification_Extension_CRLReasonCodeExtensi
                ::CA_COMPROMISE
        );
        $crlEntryExtension->setCritical(false);
```

```
        $revRequest = new
            SGCI_PKIMessage_Body_RevocationRequestMessage();

        $revRequest->addRevocationRequest($certTemplate, array(
            $crlEntryExtension));

        return $revRequest;
    }
}
```

Código Fonte C.56: Classe SGCI_PKIMessage_Body_RevocationRequestMessageTest

```
<?php
/**
 * Class that tests the
     SGCI_PKIMessage_Body_RevocationRequestContent class
 */
class SGCI_PKIMessage_Body_RevocationResponseContentTest extends
     PHPUnit_Framework_TestCase
{
    /**
     * Class being tested.
     *
     * @var SGCI_PKIMessage_Body_RevocationResponseMessage
     */
    protected $_revResponse;

    /**
     * Instantiate a
         SGCI_PKIMessage_Body_RevocationResponseContent object.
     *
     * @return void
     */
    public function setUp ()
    {
        $this->_revResponse = new
            SGCI_PKIMessage_Body_RevocationResponseContent();
    }

    /**
     * Test for loadXML
     *
     * @return void
     */
    public function testLoadXMLWithOnlyRequiredFields ()
    {
        $pkiStatusInfo = new SGCI_PKIStatusInfo();
        $pkiStatusInfo->setStatus(SGCI_PKIStatus::
            STATUS_ACCEPTED);
```

```
        $xml = $this->_getDomNode(array($pkiStatusInfo));

        $this->_revResponse->loadXML($xml);

        $status = $this->_revResponse->getStatus();
        $statusInfo = $status[0];
        $this->assertEquals(1, count($status));
        $this->assertEquals($pkiStatusInfo, $statusInfo);

        $crls = $this->_revResponse->getCrls();
        $this->assertEquals(0, count($crls));

        $revCerts = $this->_revResponse->getRevCerts();
        $this->assertEquals(1, count($revCerts));
        $this->assertNull($revCerts[0]);
    }

    /**
     * Test for loadXML
     *
     * @return void
     */
    public function testLoadXMLWithoutCrls ()
    {
        $expectedStatus = array();
        $expectedRevCerts = array();
        $pkiStatusInfo = new SGCI_PKIStatusInfo();
        $pkiStatusInfo->setStatus(SGCI_PKIStatus::
            STATUS_REJECTION);
        $expectedStatus[] = $pkiStatusInfo;
        $expectedRevCerts[] = null;

        $pkiStatusInfo = new SGCI_PKIStatusInfo();
        $pkiStatusInfo->setStatus(SGCI_PKIStatus::
            STATUS_ACCEPTED);
        $certificate = $this->_getCertificate();
        $expectedStatus[] = $pkiStatusInfo;
        $expectedRevCerts[] = $certificate;

        $xml = $this->_getDomNode($expectedStatus,
            $expectedRevCerts);

        $this->_revResponse->loadXML($xml);

        $status = $this->_revResponse->getStatus();
        $this->assertEquals(count($expectedStatus), count(
            $status));
        $this->assertEquals($expectedStatus[0], $status[0]);
        $this->assertEquals($expectedStatus[1], $status[1]);

        $revCerts = $this->_revResponse->getRevCerts();
        $this->assertEquals(count($expectedRevCerts), count(
```

```
            $revCerts));
        $this->assertEquals($expectedRevCerts[0], $revCerts[0]);
        $this->assertEquals($expectedRevCerts[1]->
            getSerialNumber(), $revCerts[1]['serialNumber']);
        $rdnSequences = $revCerts[1]['issuer']->
            getDirectoryNames();
        $this->assertEquals($expectedRevCerts[1]->getIssuer(),
            $rdnSequences[0]);

        $crls = $this->_revResponse->getCrls();
        $this->assertEquals(0, count($crls));
    }

    /**
     * Test for loadXML
     *
     * @return void
     */
    public function testLoadXMLWithAllFields ()
    {
        $pkiStatusInfo = new SGCI_PKIStatusInfo();
        $pkiStatusInfo->setStatus(SGCI_PKIStatus::
            STATUS_GRANTED_WITH_MODS);
        $certificate = $this->_getCertificate();
        $crl = $this->_getCrl();

        $xml = $this->_getDomNode(array($pkiStatusInfo), array(
            $certificate), array($crl));

        $this->_revResponse->loadXML($xml);

        $status = $this->_revResponse->getStatus();
        $this->assertEquals(1, count($status));
        $this->assertEquals($pkiStatusInfo, $status[0]);

        $crls = $this->_revResponse->getCrls();
        $this->assertEquals(1, count($crls));
        $this->assertEquals($crl, $crls[0]);

        $revCerts = $this->_revResponse->getRevCerts();
        $this->assertEquals(1, count($revCerts));
        $this->assertEquals($certificate->getSerialNumber(),
            $revCerts[0]['serialNumber']);
        $rdnSequences = $revCerts[0]['issuer']->
            getDirectoryNames();
        $this->assertEquals($certificate->getIssuer(),
            $rdnSequences[0]);
    }

    /**
     * Construct the XML from the parameters and return a
         DomNode representation of the XML
```

```
     *
     * @param array $status    The status of the revocation
           request
     * @param array $revCerts The certificate for which
           revocation was requested
     * @param array $crls      The issued crls
     *
     * @return void
     */
    protected function _getDomNode
    (
        array $status,
        array $revCerts = array(),
        array $crls = array()
    ) {
        $xml = self::getXML($status, $revCerts, $crls);

        $dom = new DOMDocument();
        $dom->loadXML($xml);

        return $dom->getElementsByTagName('RevRepContent')->item
            (0);
    }

    /**
     * Test for addRevocationResponse
     *
     * @return void
     */
    public function
        testAddRevocationResponseWithEmptyStatusInfoShouldThrowException
          ()
    {
        $this->setExpectedException('
            SGCI_Exception_InvalidParameter');

        $status = new SGCI_PKIStatusInfo();
        $this->_revResponse->addRevocationResponse($status);
    }

    /**
     * Test for addRevocationResponse
     *
     * @return void
     */
    public function
        testCanAddRevocationResponseWithoutCertificate ()
    {
        $expectedStatus = SGCI_PKIStatus::STATUS_ACCEPTED;
        $status = new SGCI_PKIStatusInfo();
        $status->setStatus($expectedStatus);
        $this->_revResponse->addRevocationResponse($status);
```

```
        $statusInfo = $this->_revResponse->getStatus();

        $revCert = $this->_revResponse->getRevCerts();

        $actualStatus = $statusInfo[0];
        $this->assertEquals($expectedStatus, $actualStatus->
            getStatus());
        $this->assertNull($revCert[0]);
    }

    /**
     * Test for addRevocationResponse
     *
     * @return void
     */
    public function testCanAddRevocationResponse ()
    {
        $expectedStatus = SGCI_PKIStatus::STATUS_ACCEPTED;
        $status = new SGCI_PKIStatusInfo();
        $status->setStatus($expectedStatus);

        $certificate = $this->_getCertificate();
        $this->_revResponse->addRevocationResponse($status,
            $certificate);

        $statusInfo = $this->_revResponse->getStatus();
        $revCert = $this->_revResponse->getRevCerts();

        $actualStatus = $statusInfo[0];

        $this->assertEquals($expectedStatus, $actualStatus->
            getStatus());

        $serialNumber = $revCert[0][
            SGCI_PKIMessage_Body_RevocationResponseContent::
            SERIAL_NUMBER];
        $this->assertEquals($certificate->getSerialNumber(),
            $serialNumber);

        $issuer = $revCert[0][
            SGCI_PKIMessage_Body_RevocationResponseContent::
            ISSUER];
        $this->assertEquals($certificate->getIssuer(), $issuer);
    }

    /**
     * Test for addCrl
     *
     * @return void
     */
    public function testCanAddCrl ()
```

```
{
    $crl = $this->_getCrl();
    $this->_revResponse->addCrl($crl);

    $crls = $this->_revResponse->getCrls();

    $this->assertEquals(1, count($crls));
    $this->assertEquals($crl->getPem(), $crls[0]->getPem());
}

/**
 * Test for addCrl
 *
 * @return void
 */
public function testCanAddTwoCrl ()
{
    $crlA = $this->_getCrl();
    $crlB = $this->_getCrl();

    $this->_revResponse->addCrl($crlA);
    $this->_revResponse->addCrl($crlB);

    $crls = $this->_revResponse->getCrls();

    $this->assertEquals(2, count($crls));
    $this->assertEquals($crlA->getPem(), $crls[0]->getPem())
        ;
    $this->assertEquals($crlB->getPem(), $crls[1]->getPem())
        ;
}

/**
 * Test for GetXMLEncoded
 *
 * @return void
 */
public function testGetXMLEncodedWithEmptyStatus()
{
    $this->setExpectedException('SGCI_Exception');

    $this->_revResponse->getXMLEncoded();
}

/**
 * Test for GetXMLEncoded
 *
 * @return void
 */
public function testGetXMLEncodedWithOnlyRequiredFields ()
{
    $pkiStatusInfo = new SGCI_PKIStatusInfo();
```

```
        $pkiStatusInfo->setStatus(SGCI_PKIStatus::STATUS_WAITING
            );

        $expectedXML = self::getXML(array($pkiStatusInfo));

        $this->_revResponse->addRevocationResponse(
            $pkiStatusInfo);

        $revRepXML = $this->_revResponse->getXMLEncoded();
        $xml = new DomDocument();
        $revRepXML = $xml->importNode($revRepXML, true);
        $xml->appendChild($revRepXML);

        $validSchema = $xml->schemaValidateSource(self::
            getRevRepMsgXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Test for GetXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithoutCrls ()
    {
        $status = array();
        $certificates = array();
        $pkiStatusInfo = new SGCI_PKIStatusInfo();
        $pkiStatusInfo->setStatus(SGCI_PKIStatus::STATUS_WAITING
            );

        $this->_revResponse->addRevocationResponse(
            $pkiStatusInfo);

        $status[] = $pkiStatusInfo;
        $certificates[] = null;

        $pkiStatusInfo = new SGCI_PKIStatusInfo();
        $pkiStatusInfo->setStatus(SGCI_PKIStatus::
            STATUS_ACCEPTED);

        $certificate = $this->_getCertificate();
        $this->_revResponse->addRevocationResponse(
            $pkiStatusInfo, $certificate);

        $status[] = $pkiStatusInfo;
        $certificates[] = $certificate;

        $expectedXML = self::getXML($status, $certificates);
```

```
        $revRepXML = $this->_revResponse->getXMLEncoded();

        $xml = new DomDocument();
        $revRepXML = $xml->importNode($revRepXML, true);
        $xml->appendChild($revRepXML);

        $validSchema = $xml->schemaValidateSource(self::
            getRevRepMsgXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Test for GetXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithAllFields()
    {
        $status = SGCI_PKIStatus::STATUS_GRANTED_WITH_MODS;
        $pkiStatusInfo = new SGCI_PKIStatusInfo();
        $pkiStatusInfo->setStatus($status);
        $certificate = $this->_getCertificate();
        $crl = $this->_getCrl();
        $expectedXML = self::getXML(array($pkiStatusInfo), array
            ($certificate), array($crl));

        $this->_revResponse->addRevocationResponse(
            $pkiStatusInfo, $certificate);
        $this->_revResponse->addCrl($crl);

        $revRepXML = $this->_revResponse->getXMLEncoded();

        $xml = new DomDocument();
        $revRepXML = $xml->importNode($revRepXML, true);
        $xml->appendChild($revRepXML);

        $validSchema = $xml->schemaValidateSource(self::
            getRevRepMsgXMLSchema());

        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Construct the XML from the parameters and return a string
          representation of the XML
     *
     * @param array  $status   The status of the revocation
          request
     * @param array  $revCerts The certificate for which
```

```
        revocation was requested
     * @param array  $crls      The issued crls
     * @param string $nodeName The name of the root node of the
        generated XML
     *
     * @return string
     */
    public static function getXML
    (
        array $status,
        array $revCerts = array(),
        array $crls = array(),
        $nodeName = 'RevRepContent'
    ) {
        $doc = new DOMDocument();

        $statusXML = '';
        foreach ($status as $statusInfo) {
            $statusInfoXML = $statusInfo->getXMLEncoded();
            $statusInfoXML = $doc->importNode($statusInfoXML,
                true);
            $statusInfoXML = $doc->saveXML($statusInfoXML);
            $statusXML .= $statusInfoXML;
        }

        $statusXML = '<status>' . $statusXML . '</status>';
        $revCertsXML = '';
        if (empty($revCerts) === false) {
            foreach ($revCerts as $revCert) {
                $issuerXML = '';
                $serialNumber = '';
                if ($revCert !== null) {
                    $dn = new
                        Labsec_Security_Certification_DataTypes_DirectoryName
                        ();
                    $dn->addDirectoryName($revCert->getIssuer())
                        ;
                    $issuerXML = $dn->getXMLEncoded('issuer');
                    $issuerXML = $doc->importNode($issuerXML,
                        true);
                    $issuerXML = $doc->saveXML($issuerXML);
                    $serialNumber = $revCert->getSerialNumber();
                    $serialNumber = '<serialNumber>' .
                        $serialNumber . '</serialNumber>';
                    $revCertsXML .= '<CertId>' . $issuerXML .
                        $serialNumber . '</CertId>';
                } else {
                    $revCertsXML .= '<CertId/>';
                }
            }
            $revCertsXML = '<revCerts>' . $revCertsXML . '</
                revCerts>';
```

```
        }
        $crlsXML = '';
        if (empty($crls) === false) {
            foreach ($crls as $crl) {
                $crlPem = $crl->getPem();
                $crlsXML .= '<CertificateList>' . $crlPem . '</
                    CertificateList>';
            }
            $crlsXML = '<crls>' . $crlsXML . '</crls>';
        }
        $xml = '<?xml version="1.0"?>' . PHP_EOL
             . '<' . $nodeName . '>'
             . $statusXML
             . $revCertsXML
             . $crlsXML
             . '</' . $nodeName . '>' . PHP_EOL;

        return $xml;
    }

    /**
     * Returns a certificate object
     *
     * @return LabsecCL_Security_Certification_CertificateCL
     */
    protected function _getCertificate ()
    {
        $data = '-----BEGIN CERTIFICATE-----'
                                                    . PHP_EOL
              . '
                  MIIB5TCCAU6gAwIBAgIBATANBgkqhkiG9w0BAQUFADAPMQ0wCwYDVQQ
                  ' . PHP_EOL
              . '
                  MB4XDTEwMDgwNjE1MzAwMloXDTIwMDgwMzE1MzAwMlowDzENMAsGA1U
                  ' . PHP_EOL
              . '
                  dDCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAnp1d1wAcIhDFD00
                  ' . PHP_EOL
              . 'j+
                  Jqtyf8aY19Xr8UYYFA98AyKrbQ9rSWu7Cun3C09yRLu4zbLDfALTUxq
                  ' . PHP_EOL
              . 'baOK5HCpo63vNT+
                  Ktkrd9Zt1cf86XWgVZI2iOwHX4jacx7QLALo2zdVmmpPDMydf
                  ' . PHP_EOL
              . 'h/bg/Kwybe08K/EZO60CAwEAAaNRME8wDwYDVR0TAQH/
                  BAUwAwEB/zAOBgNVHQ8B' . PHP_EOL
              . 'Af8EBAMCAQYwHQYDVR0OBBYEFJ8KY2/9nEEfNUep/
                  bmBfHG9/iRjMA0GA1UdEQQG' . PHP_EOL
              . '
                  MASkAjAAMA0GCSqGSIb3DQEBBQUAA4GBAH4LsmdDogNQu5Wej6rQGL8
                  ' . PHP_EOL
              . 'TugTGQo/94GpS+6vna1bkNIotTmSdNLBPHTOuvBD+
```

```
                nL9Ku3tezjmgsPV+BmblEFE' . PHP_EOL
            . 'v58L1GnapdQXfJe+
                fNcpmYZ0TmXwE12TvvL6aWFMfnGniMcbBb4QQg1c10Y2dmjN
                ' . PHP_EOL
            . 'yHP8BB85fMxn'

                . PHP_EOL
            . '-----END CERTIFICATE-----';
        return new LabsecCL_Security_Certification_CertificateCL
            ($data);
    }

    /**
     * Returns a certificate revocation list object
     *
     * @return
         LabsecCL_Security_Certification_CertificateRevocationListCL

     */
    protected function _getCrl ()
    {
        $data = '-----BEGIN X509 CRL-----'
                                                    . PHP_EOL
            . '
                MIICzzCCAbcCAQEwDQYJKoZIhvcNAQEFBQAwcTEPMA0GA1UEAxMGQUMgU1NMM
                ' . PHP_EOL
            . '
                CQYDVQQIEwJTQzELMAkGA1UEBhMCQlIxDDAKBgNVBAoTA1JOUDEPMA0GA1UEC
                ' . PHP_EOL
            . '
                SUNQRURVMQ0wCwYDVQQLEwRVRlNDMRYwFAYDVQQHEw1GbG9yaWFub3BvbGlzF
                ' . PHP_EOL
            . '
                NzA5MDYxODI1NDBaFw0wNzA5MTYxODI1NDBaMIGIMCACARIXDTA3MDkwNDE3M
                ' . PHP_EOL
            . '
                OFowDDAKBgNVHRUEAwoBADAgAgETFw0wNzA5MDQxNzEzMDhaMAwwCgYDVR0VB
                ' . PHP_EOL
            . '
                AQAwIAIBFBcNMDcwOTA0MTcxMzA4WjAMMAoGA1UdFQQDCgEAMCACARUXDTA3M
                ' . PHP_EOL
            . '
                NDE3MTMwOFowDDAKBgNVHRUEAwoBAKCBhjCBgzB1BgNVHSMEbjBsoWekZTBjM
                ' . PHP_EOL
            . '
                DgYDVQQDEwdBQyBVRlNDMQswCQYDVQQIEwJTQzELMAkGA1UEBhMCQlIxDDAKB
                ' . PHP_EOL
            . '
                BAoTA1JOUDEPMA0GA1UEChMGSUNQRURVMRYwFAYDVQQHEw1GbG9yaWFub3Bvb
                ' . PHP_EOL
            . '
                ggEEMAoGA1UdFAQDAgECMA0GCSqGSIb3DQEBBQUAA4IBAQAPSMcYaLnKcWGPj
```

```
                ' . PHP_EOL
            . 'xwa2JD/DaTttYUHL/
                cgl8WtHOtHiKlAWj6FCLdujYBktJsR29DC5pwYUr6G6nOjz
                ' . PHP_EOL
            . 'oVPBukWdZCJDpd1zJEDSIMjTAGqoOhm0vnm4mv5JkH+PMdo
                /OAs852RuYW6FCEnE' . PHP_EOL
            . '5ARWwgPErj4aaRVhtRanU65kl1sBeA6WKH4ZEHP+
                LIrp31kutCmINXJQx5qtVy/7' . PHP_EOL
            . 'RH7vKv1UdWOBvR1fu1WAeLa6shrKtyz+CKPH/0/6
                xgfP3IFAAnHFyQNEzR6V1EKq' . PHP_EOL
            . 'rgjzyklZTCbyjviltinDVnsbZUbdacsjy+
                WecahWGDHORUDMN74FYLwt1dGfAvaS' . PHP_EOL
            . 'jwj+'

                . PHP_EOL
            . '-----END X509 CRL-----';

    return new
        LabsecCL_Security_Certification_CertificateRevocationListCL
        ($data);
}

/**
 * Returns the schema validator of the CertRepMessage
     structure
 *
 * @return string
 */
public static function getRevRepMsgXMLSchema ()
{
    return '<?xml version="1.0"?>
                    <xs:schema xmlns:xs="http://www.w3.org
                        /2001/XMLSchema">

                    <xs:element name="rp" substitutionGroup
                        ="RevRepContent"/>

                    <xs:element name="RevRepContent">
                            <xs:complexType>
                                    <xs:sequence>
                                            <xs:element ref
                                                ="status"/>
                                            <xs:element ref
                                                ="revCerts"
                                                 minOccurs
                                                ="0"/>
                                            <xs:element ref
                                                ="crls"
                                                minOccurs
                                                ="0"/>
                                    </xs:sequence>
                            </xs:complexType>
```

```
</xs:element>

<xs:element name="status">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref
                            ="
                            PKIStatusInfo
                            " maxOccurs
                            ="unbounded
                            "/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="PKIStatusInfo">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name
                            ="status"
                            type="xs:
                            integer"/>
                        <xs:element name
                            ="
                            statusString
                            " type="xs:
                            string"
                            minOccurs
                            ="0"/>
                        <xs:element name
                            ="failInfo"
                             type="xs:
                            integer"
                            minOccurs
                            ="0"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="revCerts">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref
                            ="CertId"
                            maxOccurs="
                            unbounded
                            "/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="crls">
```

```
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name
                                    ="
                                    CertificateList
                                    " type="xs:
                                    string"
                                    maxOccurs="
                                    unbounded
                                    "/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>

        <xs:element name="CertId">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref
                                    ="issuer"
                                    minOccurs
                                    ="0"/>
                                <xs:element name
                                    ="
                                    serialNumber
                                    " type="xs:
                                    integer"
                                    minOccurs
                                    ="0"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>

        <xs:element name="issuer"
            substitutionGroup="GeneralName"/>

        <xs:element name="GeneralName">
                <xs:complexType>
                        <xs:choice>
                                <xs:element ref
                                    ="
                                    directoryName
                                    "/>
                        </xs:choice>
                </xs:complexType>
        </xs:element>

        <xs:element name="directoryName"
            substitutionGroup="RDNSequence"/>

        <xs:element name="RDNSequence">
                <xs:complexType>
                        <xs:sequence>
```

```
                                        <xs:choice>
                                                <xs:
                                                     element

                                                     name
                                                     ="C
                                                     "
                                                     type
                                                     ="
                                                     xs:
                                                     string
                                                     "/>

                                                     <!--

                                                     Country

                                                     -->
                                                <xs:
                                                     element

                                                     name
                                                     ="L
                                                     "
                                                     type
                                                     ="
                                                     xs:
                                                     string
                                                     "/>

                                                     <!--

                                                     Locality

                                                     -->
                                                <xs:
                                                     element

                                                     name
                                                     ="
                                                     ST"

                                                     type
                                                     ="
                                                     xs:
                                                     string
                                                     "/>

                                                     <!--

                                                     State
                                                      or
```

```
                province
                -->
            <xs:
                element
                name
                ="O
                "
                type
                ="
                xs:
                string
                "/>
                <!--
                Organization
                -->
            <xs:
                element
                name
                ="
                OU"
                type
                ="
                xs:
                string
                "/>
                <!--
                Organization
                Unit
                -->
            <xs:
                element
                name
                ="
                CN"
                type
                ="
                xs:
                string
```

```
                                                    "/>

                                                    <!--

                                                    Common

                                                    Name

                                                    -->
                                                <xs:
                                                    element

                                                    name
                                                    ="
                                                    STREET
                                                    "
                                                    type
                                                    ="
                                                    xs:
                                                    string
                                                    "/>

                                                    <!--

                                                    Street

                                                    Address

                                                    -->
                                                <xs:
                                                    element

                                                    name
                                                    ="E
                                                    "
                                                    type
                                                    ="
                                                    xs:
                                                    string
                                                    "/>

                                                    <!--
                                                     E-
                                                    mail

                                                    Address

                                                    -->
                                            </xs:choice>
                                        </xs:sequence>
                                    </xs:complexType>
                            </xs:element>
```

```
                        </xs:schema>';
    }
}
```

Código Fonte C.57: Classe SGCI_PKIMessage_Body_RevocationResponseContentTest

```
<?php
/**
 * Class that tests the
     SGCI_PKIMessage_Body_RevocationRequestMessage class
 */
class SGCI_PKIMessage_Body_RevocationResponseMessageTest extends
     PHPUnit_Framework_TestCase
{
    /**
     * test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncoded ()
    {
        $body = self::getRevocationResponseMessage();

        $expectedXML =
            SGCI_PKIMessage_Body_RevocationResponseContentTest
            ::getXML(
            $body->getStatus(), array(), array(), 'rp'
        );

        $revRepXML = $body->getXMLEncoded();
        $xml = new DomDocument();
        $revRepXML = $xml->importNode($revRepXML, true);
        $xml->appendChild($revRepXML);

        $validSchema = $xml->schemaValidateSource(
            SGCI_PKIMessage_Body_RevocationResponseContentTest::
                getRevRepMsgXMLSchema()
        );
        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Returns a SGCI_PKIMessage_Body_RevocationResponseMessage
         object
     *
     * @return SGCI_PKIMessage_Body_RevocationResponseMessage
     */
    public static function getRevocationResponseMessage ()
```

```
    {
        $pkiStatusInfo = new SGCI_PKIStatusInfo();
        $pkiStatusInfo->setStatus(SGCI_PKIStatus::
            STATUS_REJECTION);

        $revResponse = new
            SGCI_PKIMessage_Body_RevocationResponseMessage();

        $revResponse->addRevocationResponse($pkiStatusInfo);

        return $revResponse;
    }
}
```

Código Fonte C.58: Classe SGCI_PKIMessage_Body_RevocationResponseMessageTest

```
<?php
/**
 * Class that tests the
     SGCI_PKIMessage_Body_TrustedRelationshipRequest class.
 */
class SGCI_PKIMessage_Body_TrustedRelationshipRequestTest
    extends PHPUnit_Framework_TestCase
{
    /**
     * Class being tested.
     *
     * @var SGCI_PKIMessage_Body_TrustedRelationshipRequest
     */
    protected $_trustedRelReq;

    /**
     * Instantiate a
         SGCI_PKIMessage_Body_TrustedRelationshipRequest object
     *
     * @return void
     */
    public function setUp ()
    {
        $this->_trustedRelReq = new
            SGCI_PKIMessage_Body_TrustedRelationshipRequest();
    }

    /**
     * Test for loadXML
     *
     * @return void
     */
    public function testCanLoadXML ()
    {
```

```
        $expectedIpAddress = new
            Labsec_Security_Certification_DataTypes_IPAddress()
            ;
        $expectedIpAddress->setIPAddress('10.10.0.1');
        $expectedCertificate = self::getCertificate();
        $xml = $this->_getDomNode($expectedIpAddress,
            $expectedCertificate);

        $this->_trustedRelReq->loadXML($xml);

        $certificate = $this->_trustedRelReq->getCertificate();
        $this->assertEquals($expectedCertificate, $certificate);

        $iPAddress = $this->_trustedRelReq->getIpAddress();
        $this->assertEquals($expectedIpAddress, $iPAddress);
    }

    /**
     * Construct the XML from the parameters and return a
         DomNode representation of the XML
     *
     * @param Labsec_Security_Certification_DataTypes_IPAddress
         $iPAddress    The IP Address of the
     *
         requester
     * @param Labsec_Security_Certification_Certificate
         $certificate The certificate fo the
     *
         requester
     *
     * @return DOMNode
     */
    protected function _getDomNode
    (
        Labsec_Security_Certification_DataTypes_IPAddress
            $iPAddress,
        Labsec_Security_Certification_Certificate $certificate
    ) {
        $xml = $this->_getXML($iPAddress, $certificate);

        $dom = new DOMDocument();
        $dom->loadXML($xml);

        return $dom->getElementsByTagName('trr')->item(0);
    }

    /**
     * Test for setRequester
     *
     * @return void
```

```
     */
    public function testCanSetRequest ()
    {
        $expectedIpAddress = new
            Labsec_Security_Certification_DataTypes_IPAddress()
            ;
        $expectedIpAddress->setIPAddress('10.10.0.1');

        $expectedCertificate = self::getCertificate();

        $this->_trustedRelReq->setRequester($expectedIpAddress,
            $expectedCertificate);

        $certificate = $this->_trustedRelReq->getCertificate();
        $this->assertEquals($expectedCertificate, $certificate);

        $iPAddress = $this->_trustedRelReq->getIpAddress();
        $this->assertEquals($expectedIpAddress, $iPAddress);
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testGetXmlWithoutSettingRequester ()
    {
        $this->setExpectedException('SGCI_Exception');

        $this->_trustedRelReq->getXMLEncoded();
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function testCanGetXml ()
    {
        $iPAddress = new
            Labsec_Security_Certification_DataTypes_IPAddress()
            ;
        $iPAddress->setIPAddress('10.10.0.1');

        $certificate = self::getCertificate();
        $expectedXml = $this->_getXML($iPAddress, $certificate);

        $this->_trustedRelReq->setRequester($iPAddress,
            $certificate);
        $trustedRelReq = $this->_trustedRelReq->getXMLEncoded();

        $xml = new DomDocument();
```

```
        $trustedRelReq = $xml->importNode($trustedRelReq, true);
        $xml->appendChild($trustedRelReq);
        $validSchema = $xml->schemaValidateSource($this->
            _getTrustedRelReqXMLSchema());
        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXml, $xml->saveXML());
    }

    /**
     * Returns a certificate object
     *
     * @return LabsecCL_Security_Certification_CertificateCL
     */
    public static function getCertificate ()
    {
        $data = '-----BEGIN CERTIFICATE-----'
                                                . PHP_EOL
            . '
                MIIB5TCCAU6gAwIBAgIBATANBgkqhkiG9w0BAQUFADAPMQ0wCwYDVQQDE
                ' . PHP_EOL
            . '
                MB4XDTEwMDgwNjE1MzAwMloXDTIwMDgwMzE1MzAwMlowDzENMAsGA1UEA
                ' . PHP_EOL
            . '
                dDCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAnp1d1wAcIhDFD00Iq
                ' . PHP_EOL
            . 'j+
                Jqtyf8aY19Xr8UYYFA98AyKrbQ9rSWu7Cun3C09yRLu4zbLDfALTUxqGV
                ' . PHP_EOL
            . 'baOK5HCpo63vNT+
                Ktkrd9Zt1cf86XWgVZI2iOwHX4jacx7QLALo2zdVmmpPDMydf
                ' . PHP_EOL
            . 'h/bg/Kwybe08K/EZO60CAwEAAaNRME8wDwYDVR0TAQH/
                BAUwAwEB/zAOBgNVHQ8B' . PHP_EOL
            . 'Af8EBAMCAQYwHQYDVR00BBYEFJ8KY2/9nEEfNUep/bmBfHG9/
                iRjMA0GA1UdEQQG' . PHP_EOL
            . '
                MASkAjAAMA0GCSqGSIb3DQEBBQUAA4GBAH4LsmdDogNQu5Wej6rQGL8TG
                ' . PHP_EOL
            . 'TugTGQo/94GpS+6vna1bkNIotTmSdNLBPHTOuvBD+
                nL9Ku3tezjmgsPV+BmblEFE' . PHP_EOL
            . 'v58L1GnapdQXfJe+
                fNcpmYZ0TmXwE12TvvL6aWFMfnGniMcbBb4QQg1c10Y2dmjN
                ' . PHP_EOL
            . 'yHP8BB85fMxn'

                . PHP_EOL
            . '-----END CERTIFICATE-----';

        return new LabsecCL_Security_Certification_CertificateCL
            ($data);
```

```
    }

    /**
     * Construct the XML from the parameters and return a string
          representation of the XML
     *
     * @param Labsec_Security_Certification_DataTypes_IPAddress
         $iPAddress    The IP Address of the
     *
         requester
     * @param Labsec_Security_Certification_Certificate
         $certificate The certificate fo the
     *
         requester
     *
     * @return DOMNode
     */
    protected function _getXML
    (
        Labsec_Security_Certification_DataTypes_IPAddress
            $iPAddress,
        Labsec_Security_Certification_Certificate $certificate
    ) {
        $xml = '<?xml version="1.0"?>' . PHP_EOL
            . '<trr>'
            . '<iPAddress>' . $iPAddress->getIPAddress() . '</
                iPAddress>'
            . '<certificate>' . $certificate->getPem() . '</
                certificate>'
            . '</trr>' . PHP_EOL;

        return $xml;
    }

    /**
     * Returns the schema validator of the TrustedRelReq
         structure
     *
     * @return string
     */
    protected function _getTrustedRelReqXMLSchema ()
    {
        return '<?xml version="1.0"?>
                        <xs:schema xmlns:xs="http://www.w3.org
                            /2001/XMLSchema">

                        <xs:element name="trr" substitutionGroup
                            ="TrustedRelReq"/>

                        <xs:element name="TrustedRelReq">
```

```
                                    <xs:complexType>
                                            <xs:sequence>
                                                    <xs:element name
                                                        ="iPAddress
                                                        " type="xs:
                                                        string"/>
                                                    <xs:element name
                                                        ="
                                                        certificate
                                                        " type="xs:
                                                        string"/>
                                            </xs:sequence>
                                    </xs:complexType>
                            </xs:element>

                            </xs:schema>';
    }
}
```

Código Fonte C.59: Classe SGCI_PKIMessage_Body_TrustedRelationshipRequestTest

```
<?php
/**
 * Class that tests the
     SGCI_PKIMessage_Body_TrustedRelationshipResponse class.
 */
class SGCI_PKIMessage_Body_TrustedRelationshipResponseTest
    extends PHPUnit_Framework_TestCase
{
    /**
     * Class being tested.
     *
     * @var SGCI_PKIMessage_Body_TrustedRelationshipResponse
     */
    protected $_trustedRelRep;

    /**
     * Instantiate a
         SGCI_PKIMessage_Body_TrustedRelationshipResponse
         object
     *
     * @return void
     */
    public function setUp()
    {
        $this->_trustedRelRep = new
            SGCI_PKIMessage_Body_TrustedRelationshipResponse();
    }

    /**
```

```
     * Test for loadXML
     *
     * @return void
     */
    public function testCanLoadXMLWithoutCertificate ()
    {
        $expectedStatus = new SGCI_PKIStatusInfo();
        $expectedStatus->setStatus(SGCI_PKIStatus::
            STATUS_ACCEPTED);

        $xml = $this->_getDomNode($expectedStatus);

        $this->_trustedRelRep->loadXML($xml);

        $certificate = $this->_trustedRelRep->getCertificate();
        $this->assertNull($certificate);

        $status = $this->_trustedRelRep->getStatus();
        $this->assertEquals($expectedStatus, $status);
    }

    /**
     * Test for loadXML
     *
     * @return void
     */
    public function testCanLoadXMLWithAllFields ()
    {
        $expectedStatus = new SGCI_PKIStatusInfo();
        $expectedStatus->setStatus(SGCI_PKIStatus::
            STATUS_ACCEPTED);
        $expectedCertificate = $this->_getCertificate();
        $xml = $this->_getDomNode($expectedStatus,
            $expectedCertificate);

        $this->_trustedRelRep->loadXML($xml);

        $certificate = $this->_trustedRelRep->getCertificate();
        $this->assertEquals($expectedCertificate, $certificate);

        $status = $this->_trustedRelRep->getStatus();
        $this->assertEquals($expectedStatus, $status);
    }

    /**
     * Construct the XML from the parameters and return a
         DomNode representation of the XML
     *
     * @param SGCI_PKIStatusInfo                    $status
              The status of the
     *
```

```
     response
 * @param Labsec_Security_Certification_Certificate
     $certificate The certificate fo the
 *
     response
 *
 * @return DOMNode
 */
protected function _getDomNode (SGCI_PKIStatusInfo $status,
    $certificate = null)
{
    $xml = $this->_getXML($status, $certificate);

    $dom = new DOMDocument();
    $dom->loadXML($xml);

    return $dom->getElementsByTagName('trp')->item(0);
}

/**
 * Test for setStatus
 *
 * @return void
 */
public function testCanSetStatus ()
{
    $status = new SGCI_PKIStatusInfo();
    $status->setStatus(SGCI_PKIStatus::STATUS_ACCEPTED);
    $this->_trustedRelRep->setStatus($status);
    $actualStatus = $this->_trustedRelRep->getStatus();
    $this->assertEquals($status, $actualStatus);
}

/**
 * Test for setStatus
 *
 * @return void
 */
public function testSetStatusWithEmptyStatus ()
{
    $this->setExpectedException('
        SGCI_Exception_InvalidParameter');
    $status = new SGCI_PKIStatusInfo();
    $this->_trustedRelRep->setStatus($status);
}

/**
 * Test for SetCertificate
 *
 * @return void
 */
```

```
    public function
        testSetCertificateWithoutStatusSetShouldThrowException
        ()
    {
        $this->setExpectedException('SGCI_Exception');
        $this->_trustedRelRep->setCertificate($this->
            _getCertificate());
    }

    /**
     * Test for SetCertificate
     *
     * @return void
     */
    public function
        testSetCertificateWithNotApprovalStatusShouldThrowException
         ()
    {
        $this->setExpectedException('SGCI_Exception');
        $status = new SGCI_PKIStatusInfo();
        $status->setStatus(SGCI_PKIStatus::STATUS_REJECTION);
        $this->_trustedRelRep->setStatus($status);
        $this->_trustedRelRep->setCertificate($this->
            _getCertificate());
    }

    /**
     * Test for SetCertificate
     *
     * @return void
     */
    public function testCanSetCertificate ()
    {
        $status = new SGCI_PKIStatusInfo();
        $status->setStatus(SGCI_PKIStatus::STATUS_ACCEPTED);
        $this->_trustedRelRep->setStatus($status);
        $certificate = $this->_getCertificate();
        $this->_trustedRelRep->setCertificate($certificate);
        $actualCertificate = $this->_trustedRelRep->
            getCertificate();
        $this->assertEquals($certificate, $actualCertificate);
    }

    /**
     * Test for getXMLEncoded
     *
     * @return void
     */
    public function
        testGetXmlWithoutSettingStatusShouldThrowException ()
    {
        $this->setExpectedException('SGCI_Exception');
```

```
        $this->_trustedRelRep->getXMLEncoded();
    }

    /**
     * Test for GetXMLEncoded
     *
     * @return void
     */
    public function
        testGetXMLEncodedWithoutStatusShouldThrowException ()
    {
        $this->setExpectedException('SGCI_Exception');
        $this->_trustedRelRep->getXMLEncoded();
    }

    /**
     * Test for GetXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithOnlyRequiredFields ()
    {
        $pkiStatusInfo = new SGCI_PKIStatusInfo();
        $pkiStatusInfo->setStatus(SGCI_PKIStatus::
            STATUS_REJECTION);
        $expectedXML = $this->_getXML($pkiStatusInfo);
        $this->_trustedRelRep->setStatus($pkiStatusInfo);

        $trustedRelRepXML = $this->_trustedRelRep->getXMLEncoded
            ();

        $xml = new DomDocument();
        $trustedRelRepXML = $xml->importNode($trustedRelRepXML,
            true);
        $xml->appendChild($trustedRelRepXML);

        $validSchema = $xml->schemaValidateSource($this->
            _getTrustedRelRepXmlSchema());
        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Test for GetXMLEncoded
     *
     * @return void
     */
    public function testGetXMLEncodedWithAllFields ()
    {
        $certificate = $this->_getCertificate();
        $pkiStatusInfo = new SGCI_PKIStatusInfo();
```

```
        $pkiStatusInfo->setStatus(SGCI_PKIStatus::
            STATUS_ACCEPTED);
        $expectedXML = $this->_getXML($pkiStatusInfo,
            $certificate);

        $this->_trustedRelRep->setStatus($pkiStatusInfo);
        $this->_trustedRelRep->setCertificate($certificate);

        $trustedRelRepXML = $this->_trustedRelRep->getXMLEncoded
            ();

        $xml = new DomDocument();
        $trustedRelRepXML = $xml->importNode($trustedRelRepXML,
            true);
        $xml->appendChild($trustedRelRepXML);

        $validSchema = $xml->schemaValidateSource($this->
            _getTrustedRelRepXmlSchema());
        $this->assertTrue($validSchema);
        $this->assertEquals($expectedXML, $xml->saveXML());
    }

    /**
     * Returns a certificate object
     *
     * @return LabsecCL_Security_Certification_CertificateCL
     */
    protected function _getCertificate()
    {
        $data = '-----BEGIN CERTIFICATE-----'
                                                . PHP_EOL
            . '
                MIIB5TCCAU6gAwIBAgIBATANBgkqhkiG9w0BAQUFADAPMQ0wCwYDVQQDEwRUZXN
                ' . PHP_EOL
            . '
                MB4XDTEwMDgwNjE1MzAwMloXDTIwMDgwMzE1MzAwMlowDzENMAsGA1UEAxMEVGV
                ' . PHP_EOL
            . '
                dDCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAnp1d1wAcIhDFD00IqQyqFmk
                ' . PHP_EOL
            . 'j+
                Jqtyf8aY19Xr8UYYFA98AyKrbQ9rSWu7Cun3C09yRLu4zbLDfALTUxqGVVSQB7
                ' . PHP_EOL
            . 'baOK5HCpo63vNT+
                Ktkrd9Zt1cf86XWgVZI2iOwHX4jacx7QLALo2zdVmmpPDMydf
                ' . PHP_EOL
            . 'h/bg/Kwybe08K/EZO60CAwEAAaNRME8wDwYDVR0TAQH/
                BAUwAwEB/zAOBgNVHQ8B' . PHP_EOL
            . 'Af8EBAMCAQYwHQYDVR00BBYEFJ8KY2/9nEEfNUep/bmBfHG9/
                iRjMA0GA1UdEQQG' . PHP_EOL
            . '
                MASkAjAAMA0GCSqGSIb3DQEBBQUAA4GBAH4LsmdDogNQu5Wej6rQGL8TGgq4kpH
```

```
            ' . PHP_EOL
        . 'TugTGQo/94GpS+6vna1bkNIotTmSdNLBPHTOuvBD+
            nL9Ku3tezjmgsPV+BmblEFE' . PHP_EOL
        . 'v58L1GnapdQXfJe+
            fNcpmYZ0TmXwE12TvvL6aWFMfnGniMcbBb4QQg1c10Y2dmjN
            ' . PHP_EOL
        . 'yHP8BB85fMxn'

            . PHP_EOL
        . '-----END CERTIFICATE-----';

    return new LabsecCL_Security_Certification_CertificateCL
        ($data);
}

/**
 * Construct the XML from the parameters and return a string
      representation of the XML
 *
 * @param SGCI_PKIStatusInfo                          $status
        The status of the
 *

     response
 * @param Labsec_Security_Certification_Certificate
     $certificate The certificate fo the
 *

     response
 *
 * @return DOMNode
 */
protected function _getXML(SGCI_PKIStatusInfo $status,
    $certificate = null)
{
    $statusXml = $status->getXMLEncoded('status');
    $doc = new DOMDocument();
    $statusXml = $doc->importNode($statusXml, true);
    $statusXml = $doc->saveXML($statusXml);

    $certificateXml = '';
    if ($certificate !== null) {
        $certificateXml = '<certificate>' . $certificate->
            getPem() . '</certificate>';
    }
    $xml = '<?xml version="1.0"?>' . PHP_EOL
        . '<trp>'
        . $statusXml
        . $certificateXml
        . '</trp>' . PHP_EOL;

    return $xml;
```

```
    }

    /**
     * Returns the schema validator of the TrustedRelReq
         structure
     *
     * @return string
     */
    protected function _getTrustedRelRepXMLSchema ()
    {
        return '<?xml version="1.0"?>
                        <xs:schema xmlns:xs="http://www.w3.org
                            /2001/XMLSchema">

                        <xs:element name="trp" substitutionGroup
                            ="TrustedRelRep"/>

                        <xs:element name="TrustedRelRep">
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element ref
                                                    ="status"/>
                                                <xs:element name
                                                    ="
                                                    certificate
                                                    " type="xs:
                                                    string"
                                                    minOccurs
                                                    ="0"/>
                                        </xs:sequence>
                                </xs:complexType>
                        </xs:element>

                        <xs:element name="status"
                            substitutionGroup="PKIStatusInfo"/>

                        <xs:element name="PKIStatusInfo">
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element name
                                                    ="status"
                                                    type="xs:
                                                    integer"/>
                                                <xs:element name
                                                    ="
                                                    statusString
                                                    " type="xs:
                                                    string"
                                                    minOccurs
                                                    ="0"/>
                                                <xs:element name
                                                    ="failInfo"
```

```
                                                                type="xs:
                                                               integer"
                                                               minOccurs
                                                               ="0"/>
                                                        </xs:sequence>
                                                </xs:complexType>
                                        </xs:element>

                                        </xs:schema>';
    }
}
```

Código Fonte C.60: Classe SGCI_PKIMessage_Body_TrustedRelationshipResponseTest

APÊNDICE D – Artigo

Análise e implementação de um protocolo de gerenciamento de certificados

Anderson Luiz Silvério[1]

[1]Laboratório de Segurança em Computação (LabSEC)
Departamento de Informática e Estatística (INE)
Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

anderson.luiz@inf.ufsc.br

Abstract. *This meta-paper describes the style to be used in articles and short papers for SBC conferences. For papers in English, you should add just an abstract while for the papers in Portuguese, we also ask for an abstract in Portuguese ("resumo"). In both cases, abstracts should not have more than 10 lines and must be in the first page of the paper.*

Resumo. *Este trabalho tem como objetivo estudar e implementar um protocolo para a comunicação entre Autoridades Certificadoras e de Registro para o Sistema de Gerenciamento de Certificados Digitais da Infraestrutura de Chaves Públicas para Ensino e Pesquisa (SGCI). A versão 1.3.7 do SGCI possui um protocolo próprio para fazer a comunicação entre ACs e ARs, que não garante a integridade e autenticidade das mensagens trocadas entre AC e AR. Este trabalho propõe a implementação de um protocolo baseado no Certificate Management Protocol (CMP). Para tanto, foram analisadas e implementadas as mudanças necessárias ao SGCI para suportar tal protocolo, além do desenvolvimento de uma biblioteca para a geração e manipulação das mensagens do CMP em XML.*

1. Introdução

A sociedade moderna está convergindo para o uso de meios eletrônicos e *online* para a realização de tarefas cotidianas. Por exemplo, o uso do *internet banking* para pagar contas ao invés de ir pessoalmente ao banco. Com isso é possível reduzir tempo e custos gastos nestas tarefas. Por outro lado, na ausência de contato visual entre as partes envolvidas em sistemas *online*, há a necessidade de mecanismos de autenticação e segurança que devem garantir a integridade dos dados compartilhados entre usuário e sistema, autenticar as partes envolvidas, entre outros requisitos. A certificação digital tem sido utilizada de modo a satisfazer tais requisitos.

Certificados digitais ligam uma pessoa ou entidade à um par de chaves. Com o par de chaves é possível realizar operações, como a assinatura digital, que garantem a autenticidade, integridade, confidencialidade e irretratabilidade. Para a emissão de certificados digitais é necessário a presença de uma entidade confiável, chamada Autoridade Certificadora (AC), para certificar que as informações presentes num certificado digital pertencem a determinada pessoa.

As ACs normalmente delegam a função de fazer a interação com o requerente do certificado à Autoridades de Registro (ARs). A AR tem o papel de receber o pedido de requisição do requerente do certificado, verificar e validar se os dados presentes na requisição estão de acordo com os dados do requerente. Após a validação dos dados, a AR encaminha a requisição à AC que então pode emitir o certificado. Em seguida a AC envia um resposta à AR, contendo o certificado, para que a AR possa encaminhar o certificado ao requerente.

A comunicação entre as partes envolvidas pode ser feita de várias formas, via e-mail por exemplo. Contudo ela envolve sempre os mesmo objetivos, emitir ou revogar certificados digitais. Existem protocolos padronizados que definem as mensagens que necessitam ser trocadas entre AC e AR durante o processo de requisição ou revogação de certificados. Todavia observa-se que muitos *softwares* utilizados para gerenciar ACs e ARs implementam o seu próprio protocolo, causando incompatibilidades com entidades que utilizem outros *softwares*.

Nas seções 2 e 3 é apresentada uma breve revisão bibliográfica sobre o Sistema de Gerenciamento de Certificados Digitais ICPEDU (SGCI) e o *Certificate Management Protocol* (CMP), respectivamente. A seção 4 apresenta como o CMP é utilizado neste trabalho e a sua implementação, a seção 5 apresenta a proposta deste trabalho para a distribuição do par de chaves de ACs e ARs e a seção 6 aborda as constribuições deste trabalho para o SGCI. Por fim, as considerações finais são realizadas na seção 7.

2. Sistema de Gerenciamento de Certificados Digitais ICPEDU

O Sistema de Gerenciamento de Certificados Digitais da Infraestrutura de Chaves Públicas para Ensino e Pesquisa (SGCI) é um SGC desenvolvido para o âmbito acadêmico, fazendo parte do projeto ICPEDU, financiado pela Rede Nacional de Ensino e Pesquisa (RNP). Ele foi concebido para utilizar somente software livre, ser flexível, simples e robusto, provendo as funcionalidades necessárias para o gerenciamento de ICPs.

Este software é divido em cinco módulos, descrito a seguir:

- Criador: Responsável pelas configurações do software e disponibilização de entidades e usuários. Neste módulo é possível cadastrar e deletar ACs, ARs e usuários, além de vincular usuários com as entidades;
- Administrador de AC: Responsável pela administração de autoridades certificadoras. Neste módulo é possível cadastrar e deletar operadores, cadastrar modelos de certificados, gerenciar relacionamentos de confiança e definir o período para emissão de LCRs;
- Administrador de AR: Responsável pela administração de autoridades de registro. Neste módulo é possível cadastrar e deletar operadores e gerenciar relacionamentos de confiança;
- Operador de AC: Responsável pela operação de autoridades certificadoras. Neste módulo é possível aprovar ou rejeitar requisições de certificado e de revogação, emitindo os certificados para as requisições aprovadas, e emitir LCRs;
- Operador de AR: Responsável pela operação de autoridades de registro. Neste módulo é possível aprovar ou rejeitar requisições de certificado e de revogação, encaminhando as requisições aprovadas para a AC responsável.

Atualmente o Laboratório de Segurança em Computação (LabSEC) é responsável pelo desenvolvimento e manutenção do SGCI que possui uma versão estável, 1.3.7, e uma nova versão, 2.0.0, encontra-se em desenvolvimento.

3. Certificate Management Protocol

O Certificate Management Protocol (CMP) [Adams et al. 2005] é um protocolo utilizado para criação e gerenciamento de certificados digitais X.509v3 [Cooper et al. 2008] e define mensagens que permitem a interação online de diferentes componentes de uma ICP.

Toda mensagem definida pelo CMP possui uma estrutura básica, contendo os seguintes campos:

- cabeçalho: Apresenta informações comuns a várias mensagens, utilizadas para identificar o emissor e destinatário, por exemplo;
- corpo: Apresenta informações específicas para cada requisição;
- proteção: Contém bits que protegem a mensagem. Por exemplo, a assinatura dos campos citados acima. Este campo é opcional;
- certificados extras: Pode ser usado para carregar certificados necessários por uma das partes. Este campo é opcional.

O transporte das mensagens do CMP é feito sobre o protocolo HTTP e é definido pelo CMPTrans [Kause and Peylo 2011].

4. Implementação do protocolo

Para a representação das mensagens definidas pelo CMP optou-se por utilizar o formato XML, por ser amplamente utilizado atualmente para o compartilhamento de informações, além de ser de código aberto e independente de plataforma. A conversão das estruturas ASN.1 descritas na RFC4210 para XML foram feitas utilizando as regras de codificação XML Enconding Rules (XER) [ITU-T 2001].

4.1. Validação do XML

Um documento XML por si só não possui informações suficientes para descrever a estrutura que o documento deverá ter. Existem diversas tecnologias que possibilitam definir uma estrutura para documentos XML, as duas principais e recomendadas pela W3C são DTD e XSD. Para este trabalho foi escolhida a XSD, por ser mais expressiva em relação a DTD, simples, e usar a sintaxe do XML.

O trecho de código 1 mostra a definição de estrutura PKIMessage em ASN.1. Não existe um documento especificando como fazer a conversão das estruturas ASN.1 para XSD, então a conversão foi feita visando manter a mesma semântica entre as estruturas ASN.1 e XSD.

```
PKIMessage ::= SEQUENCE {
    header          PKIHeader,
    body            PKIBody,
    protection   [0] PKIProtection OPTIONAL,
    extraCerts   [1] SEQUENCE SIZE (1..MAX) OF CMPCertificate
                     OPTIONAL
}
```

```
PKIHeader ::= SEQUENCE { ... }
PKIBody ::= CHOICE { ... }
PKIProtection ::= BIT STRING
CMPCertificate ::= CHOICE { ... }
```

Código Fonte 1. Exemplo de definição da estrutura PKIMessage em ASN.1

Grande parte das definições encontradas em ASN.1 têm um correspondente direto em XSD, porém em alguns casos foi necessário utilizar uma forma alternativa para manter o mesmo efeito. As principais estruturas convertidas que merecem destaque são:

- Estruturas: Estruturas complexas em ASN.1, podem ser declaradas em XSD através da tag *<xs:complexType>*
- Sequências: O elemento SEQUENCE em ASN.1 corresponde à tag *<xs:sequence>*
- Escolhas: O elemento CHOICE em ASN.1 corresponde à tag *<xs:choice>*
- Campos Opcionais: O atributo OPTIONAL em ASN.1 não tem um correspondente direto em XSD. Para conseguir este efeito, foi utilizado o atributo "minOccurs" com o valor igual a zero, especificando que o elemento pode aparecer zero vezes, ou seja, é opcional.
- Sequências de tamanho indefinido: O atributo SEQUENCE SIZE (1..MAX) não tem um correspondente direto em XSD. Para este caso, foi utilizado o atributo maxOccurs, com o valor igual a "unbounded", na tag *<xs:sequence>*. O valor "unbounded" significa que a sequência pode ter tamanho infinito. Para sequências de tamanhos fixos, utiliza-se o atributo "minOccurs" com o valor mínimo e "maxOccurs" com o valor máximo.

Aplicando as regras citadas acima para fazer a conversão, obteve-se o XSD do trecho de código 2, correspondente a mesma estrutura definida em ASN.1 no trecho de código 1.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="PKIMessage">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="header"/>
                        <xs:element ref="body"/>
            <xs:element ref="protection" minOccurs="0"/>
            <xs:element ref="extraCerts"/>
                 </xs:sequence>
         </xs:complexType>
</xs:element>

<xs:element name="header" substitutionGroup="PKIHeader"/>
<xs:element name="body" substitutionGroup="PKIBody"/>
<xs:element name="protection" substitutionGroup="PKIProtection"/>
<xs:element name="extraCerts">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="CMPCertificate" minOccurs="0"
                            maxOccurs="unbounded"/>
```

```
                </xs:sequence>
            </xs:complexType>
</xs:element>
</xs:schema>
```

Código Fonte 2. Exemplo de definição da estrutura PKIMessage em XSD

4.2. Biblioteca criptográfica

Inicialmente foi feito um levantamento na literatura das bibliotecas já existentes que suportam o CMP. Foram encontradas duas bibliotecas, a *cryptLib* [Digital Data Security Limited 2011] e a *cmpForOpenssl* [Martin Peylo 2011]. Foi desconsiderado o uso destas bibliotecas para este trabalho pelos seguintes motivos:

- são escritas na linguagem C. Desta forma seria ainda necessário portar as funcionalidades para o PHP, de modo a utilizar com o SGCI;
- não possuem suporte a XML.

Não existindo nenhuma biblioteca que satisfaça as necessidades deste trabalho, foi criada uma nova biblioteca, orientada a objetos e em PHP. Um dos objetivos desta biblioteca é fornecer uma interface simples e independente, podendo ser utilizada por diferentes softwares. Além disso, ela é facilmente extensível, possibilitando adicionar as mensagens não tratadas por este trabalho ou fazer implementações customizadas das mensagens.

5. Relacionamento de confiança

O CMP não provê meios para a distribuição da chave pública de entidades, não sendo possível garantir que dada mensagem foi gerada por determinada entidade. Para isso foi necessário estender o CMP, adicionando quatro novas mensagens ao *PKIBody*, para possibilitar o estabelecimento de relações de confiança. Relação de confiança é um conceito utilizado pelo SGCI, para uma AC informar em quais ARs ela confia e aceita receber requisições e para uma AR informar para quais ACs ela pode enviar requisições e receber respostas.

Para as ACs também foi necessário gerar um novo par de chaves e um certificado para distribuir a chave pública deste novo par de chaves, chamado de certificado de transporte. A nova chave privada é utilizada para assinar as mensagens do CMP, devido ao fato de as chaves privadas das ACs terem seu uso muito restrito, devendo ser utilizado somente para assinar certificados e LCRs. O novo par de chaves e o certificado são utilizados exclusivamente para a geração e verificação das assinaturas das mensagens do CMP.

A figura 1 mostra como funciona o estabelecimento de uma relação de confiança no SGCI. Primeiro a AR envia uma mensagem ao SGCI para listar todas as entidades cadastradas. O SGCI retorna uma lista com os certificados das entidades cadastradas e a AR escolhe com qual das entidades deseja estabelecer relacionamento de confiança. Em seguida a AR envia à entidade escolhida um pedido para o estabelecimento de relação de confiança. Neste ponto a AC precisa aprovar ou rejeitar o pedido feito pela AR e enviar uma resposta para a AR. A requisição e a resposta são assinadas com a chave privada da AR e da AC, respectivamente.

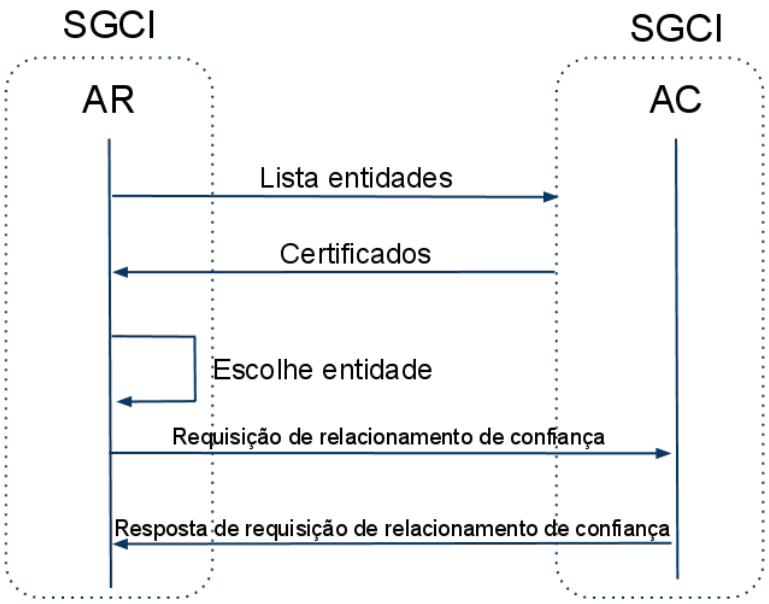


Figura 1. Estabelecimento de relacionamento de confiança no SGCI

A mensagem destinada a listar os certificados das entidades é uma tag XML vazia, apresentada no trecho de código 3. A resposta desta mensagem é uma lista dos certificados das entidades cadastradas. Se nenhuma entidade estiver cadastrada, é retornada a tag XML vazia. O trecho de código 4 apresenta a estrutura ListCertsRep em XML.

```
<ListCertsReq></ListCertsReq>
```

Código Fonte 3. Estrutura ListCertsReq em XML

```
<ListCertsRep>
    <certificate></certificate>
</ListCertsRep>
```

Código Fonte 4. Estrutura ListCertsRep em XML

Para minimizar o tráfego dos certificados pela rede é possível fazer uma otimização da mensagem ListCertsReq, incluindo, por exemplo, um template de certificado, contendo informações sobre as entidades que se deseja listar. Porém esta otimização só faria sentido em um sistema com centenas ou milhares de entidades cadastradas. Como cada instância do SGCI usualmente contém apenas algumas entidades cadastradas, tal otimização traria um maior processamento para o SGCI que não resultaria em uma redução significativa nos dados trafegados pela rede. Por este motivo não foi implementado neste

trabalho.

A requisição de relacionamento de confiança contém, no PKIBody, a estrutura TrustedRelReq, apresentada no trecho de código 5. Ela contém o endereço IP, o certificado da entidade requisitante e o certificado de transporte. O certificado de transporte só deverá estar presente se a requisição for gerada por uma AC. Neste caso este certificado será utilizado pela AR para verificar a assinatura das mensagens subsequentes entre as entidades.

```
<TrustedRelReq>
   <iPAddress></iPAddress>
   <certificate></certificate>
   <transportCertificate></transportCertificate> <!— Opcional —>
</TrustedRelReq>
```

Código Fonte 5. Estrutura TrustedRelReq em XML

A resposta para a requisição de relacionamento de confiança contém, no PKIBody, a estrutura TrustedRelRep, apresentada no trecho de código 6.

Ela contém o status do pedido, descrito pela estrutura PKIStatusInfo do CMP, o certificado da entidade e o certificado de transporte. Ambos os certificado são opcionais e só deverão estar presentes caso a relação de confiança seja aprovada. E o certificado de transporte só deverá estar presente se a resposta for gerada por uma AC, pois a AR utiliza a sua própria chave privada para assinar as mensagens subsequentes que serão trocadas entre as entidades.

```
<TrustedRelRep>
   <status></status> <!— PKIStatusInfo —>
   <certificate></certificate> <!— Opcional —>
   <transportCertificate></transportCertificate> <!— Opcional —>
</TrustedRelRep>
```

Código Fonte 6. Estrutura TrustedRelRep em XML

6. Contribuições ao SGCI

Na atual versão do SGCI, 1.3.7, a comunicação entre as entidades é feita apenas de forma manual. Neste modelo, o operador da AR importa a requisição de certificado, aprova a requisição e ela é disponibilizada ao operador para download. O operador então envia a requisição para a AC de forma manual, através e-mail por exemplo. Na AC, o operador deve importar a requisição recebida pela AR, aprovar ou rejeitar a requisição e enviar a resposta para a AR também de forma manual.

A partir deste trabalho, o SGCI passa a ter dois novos modelos de comunicação, ambos online. O primeiro modelo é conhecido como modelo online com AC de resposta manual, cuja única diferença do modelo offline é que o envio das mensagens entre a AR e a AC é feita de forma automática.

No segundo modelo, conhecido como modelo online com AC de resposta automática, além do envio das mensagens ser feito de forma automática, a aprovação das requisições enviadas pela AR na AC também é feita de forma automática.

7. Considerações Finais

Este trabalho teve como objetivo estudar e analisar protocolos para a comunicação entre Autoridades Certificadoras e de Registro presentes na literatura, além da implementação de um protocolo para o Sistema de Gerenciamento de Certificados da Infraestrutura de Chaves Públicas para Ensino e Pesquisa (SGCI). O protocolo escolhido para implementar baseia-se no CMP e possui as seguintes funcionalidades: requisição para emissão e revogação de certificados; requisição para estabelecimento de relacionamento de confiança; e *pollings* para obter o estado das requisições pendentes. Ele foi escolhido, pois o CMP possui uma variedade maior de operações que podem ser realizadas, além de não depender de implementações externas.

Neste trabalho foi proposto o uso de XML para codificar as mensagens do CMP, pois o XML é uma linguagem amplamente utilizada para a troca de informações entre diferentes sistemas, além de ser simples de implementar e fornecer uma representação textual, legível para humanos. Na seção 4 foi descrito como foi feito o mapeamento das estruturas ASN.1 utilizadas neste trabalho para XML. Além do mapeamento das estruturas para XML, foram criadas regras, utilizando a linguagem XSD, para verificar se dado arquivo XML representa corretamente uma mensagem do CMP suportada pela implementação deste trabalho.

Também foi implementada uma biblioteca, para facilitar a geração e manipulação das mensagens do CMP, na linguagem PHP, orientada a objetos, seguindo as práticas de programação de Desenvolvimento Orientado a Testes (TDD, do inglês *Test Driven Development*) e *Clean Code*, a fim de garantir uma alta qualidade no código desenvolvido.

Na seção 5 foi proposta uma forma para a distribuição do par de chaves entre autoridades certificadoras e de registro, chamada de relacionamento de confiança. Esta proposta consiste em adiconar novas mensagens ao CMP para fazer o estabelecimento de relação de confiança entre AC e AR, além da utilização de um novo par de chaves para fazer a assinatura das mensagens a fim de preservar a chave privada de ACs e ARs. Por fim, na seção 2 foram apresentadas as contribuições destre trabalho para o SGCI.

Com as principais estruturas do CMP e a camada para o transporte implementadas, propõe-se a implementação das operações suportadas pelo CMP que não foram abordadas neste trabalho, como a substituição do par de chaves de ACs, e a integração das mesmas ao SGCI. A implementação destas operações aumentaria a flexibilidade do *software*, além de tornar as operações mais transparentes ao usuário final.

Outro trabalho futuro proposto, relacionado com as limitações deste trabalho, é o estudo de como estender o protocolo implementado para realizar a comunicação entre usuário final e AR, através da implementação de uma interface pública, chamada módulo público. A existência de um módulo público no SGCI automatizaria a comunicação entre o requerente do certificado e a AR.

Propõe-se ainda a formalização das estruturas adicionadas ao CMP e do uso de XML para a codificação das mensagens, não previsto pelo CMP. Tais formalizações são importantes para garantir que as alterações feitas não afetam o funcionamento do CMP e a interoperabilidade com outros sistemas que utilizem o CMP.

Por fim, propõe-se o estudo de como aumentar a segurança das chaves privadas

das entidades *online*. Quando uma entidade é configurada para funcionar de forma *online*, sua chave privada necessita ficar disponível para o uso do SGCI. Este é um problema conhecido na literatura, porém ainda sem solução. Algumas soluções sugeridas é o uso de *tokens* criptográficos ou de um *Trusted Platform Module* (TPM).

Referências

Adams, C., Farrell, S., Kause, T., and Mononen, T. (2005). Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP). RFC 4210 (Proposed Standard).

Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and Polk, W. (2008). Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard).

Digital Data Security Limited (2011). Cryptlib security software.

ITU-T (2001). Information technology – asn.1 encoding rules: Xml encoding rules (xer). Recommendation X.693, International Telecommunication Union.

Kause, T. and Peylo, M. (2011). Internet X.509 Public Key Infrastructure – Transport Protocols for CMP.

Martin Peylo (2011). Cmp for openssl.

ANEXO A – Definição ASN.1 das estruturas das mensagens do CMP

```
PKIMessage ::= SEQUENCE {
    header              PKIHeader,
    body                PKIBody,
    protection      [0] PKIProtection OPTIONAL,
    extraCerts      [1] SEQUENCE SIZE (1..MAX) OF CMPCertificate
                        OPTIONAL
}
```

Código Fonte A.1: Estrutura PKIMessage em ASN.1

```
PKIHeader ::= SEQUENCE {
    pvno                INTEGER     { cmp1999(1), cmp2000(2) },
    sender              GeneralName,
    recipient           GeneralName,
    messageTime     [0] GeneralizedTime         OPTIONAL,
    protectionAlg   [1] AlgorithmIdentifier     OPTIONAL,
    senderKID       [2] KeyIdentifier           OPTIONAL,
    recipKID        [3] KeyIdentifier           OPTIONAL,
    transactionID   [4] OCTET STRING            OPTIONAL,
    senderNonce     [5] OCTET STRING            OPTIONAL,
    recipNonce      [6] OCTET STRING            OPTIONAL,
    freeText        [7] PKIFreeText             OPTIONAL,
    generalInfo     [8] SEQUENCE SIZE (1..MAX) OF
                        InfoTypeAndValue        OPTIONAL
}
PKIFreeText ::= SEQUENCE SIZE (1..MAX) OF UTF8String
```

Código Fonte A.2: Estrutura PKIHeader em ASN.1

```
PKIBody ::= CHOICE {
    ir          [0]  CertReqMessages,        --Initialization Req
    ip          [1]  CertRepMessage,         --Initialization Resp
    cr          [2]  CertReqMessages,        --Certification Req
    cp          [3]  CertRepMessage,         --Certification Resp
    p10cr       [4]  CertificationRequest,   --PKCS #10 Cert. Req.
    popdecc     [5]  POPODecKeyChallContent --pop Challenge
    popdecr     [6]  POPODecKeyRespContent, --pop Response
    kur         [7]  CertReqMessages,        --Key Update Request
    kup         [8]  CertRepMessage,         --Key Update Response
    krr         [9]  CertReqMessages,        --Key Recovery Req
    krp         [10] KeyRecRepContent,       --Key Recovery Resp
    rr          [11] RevReqContent,          --Revocation Request
    rp          [12] RevRepContent,          --Revocation Response
    ccr         [13] CertReqMessages,        --Cross-Cert. Request
    ccp         [14] CertRepMessage,         --Cross-Cert. Resp
    ckuann      [15] CAKeyUpdAnnContent,     --CA Key Update Ann.
    cann        [16] CertAnnContent,         --Certificate Ann.
    rann        [17] RevAnnContent,          --Revocation Ann.
    crlann      [18] CRLAnnContent,          --CRL Announcement
    pkiconf     [19] PKIConfirmContent,      --Confirmation
```

```
    nested    [20] NestedMessageContent,   --Nested Message
    genm      [21] GenMsgContent,          --General Message
    genp      [22] GenRepContent,          --General Response
    error     [23] ErrorMsgContent,        --Error Message
    certConf  [24] CertConfirmContent,     --Certificate confirm
    pollReq   [25] PollReqContent,         --Polling request
    pollRep   [26] PollRepContent          --Polling response
}
```

Código Fonte A.3: Estrutura PKIBody em ASN.1

```
PKIStatus ::= INTEGER {
        accepted                (0),
        grantedWithMods         (1),
        rejection               (2),
        waiting                 (3),
        revocationWarning       (4),
        revocationNotification  (5),
        keyUpdateWarning        (6)
}
```

Código Fonte A.4: Estrutura PKIStatus em ASN.1

```
PKIFailureInfo ::= BIT STRING {
        badAlg              (0),
        badMessageCheck     (1),
        badRequest          (2),
        badTime             (3),
        badCertId           (4),
        badDataFormat       (5),
        wrongAuthority      (6),
        incorrectData       (7),
        missingTimeStamp    (8),
        badPOP              (9),
        certRevoked         (10),
        certConfirmed       (11),
        wrongIntegrity      (12),
        badRecipientNonce   (13),
        timeNotAvailable    (14),
        unacceptedPolicy    (15),
        unacceptedExtension (16),
        addInfoNotAvailable (17),
        badSenderNonce      (18),
        badCertTemplate     (19),
        signerNotTrusted    (20),
        transactionIdInUse  (21),
        unsupportedVersion  (22),
        notAuthorized       (23),
        systemUnavail       (24),
        systemFailure       (25),
        duplicateCertReq    (26)
```

```
}
```

Código Fonte A.5: Estrutura PKIFailureInfo em ASN.1

```
PKIStatusInfo ::= SEQUENCE {
      status          PKIStatus,
      statusString    PKIFreeText     OPTIONAL,
      failInfo        PKIFailureInfo  OPTIONAL
}
```

Código Fonte A.6: Estrutura PKIStatusInfo em ASN.1

```
CertReqMessages ::= SEQUENCE SIZE (1..MAX) OF CertReqMsg

CertReqMsg ::= SEQUENCE {
   certReq    CertRequest,
   popo        ProofOfPossession  OPTIONAL,
   -- content depends upon key type
   regInfo    SEQUENCE SIZE(1..MAX) of AttributeTypeAndValue
       OPTIONAL
}

CertRequest ::= SEQUENCE {
   certReqId     INTEGER,          -- ID for matching request and
       reply
   certTemplate  CertTemplate, --Selected fields of cert to be
       issued
   controls      Controls OPTIONAL -- Attributes affecting
       issuance
}
```

Código Fonte A.7: Estrutura CertReqMessages em ASN.1

```
CertTemplate ::= SEQUENCE {
   version      [0] Version                 OPTIONAL,
   serialNumber [1] INTEGER                 OPTIONAL,
   signingAlg   [2] AlgorithmIdentifier     OPTIONAL,
   issuer       [3] Name                    OPTIONAL,
   validity     [4] OptionalValidity        OPTIONAL,
   subject      [5] Name                    OPTIONAL,
   publicKey    [6] SubjectPublicKeyInfo    OPTIONAL,
   issuerUID    [7] UniqueIdentifier        OPTIONAL,
   subjectUID   [8] UniqueIdentifier        OPTIONAL,
   extensions   [9] Extensions              OPTIONAL
}

OptionalValidity ::= SEQUENCE {
   notBefore  [0] Time OPTIONAL,
   notAfter   [1] Time OPTIONAL
} --at least one must be present
```

```
Time ::= CHOICE {
   utcTime          UTCTime,
   generalTime      GeneralizedTime
}
```

Código Fonte A.8: Estrutura CertTemplate em ASN.1

```
CertRepMessage ::= SEQUENCE {
    caPubs           [1] SEQUENCE SIZE (1..MAX) OF Certificate
                         OPTIONAL,
    response             SEQUENCE OF CertResponse
}

CertResponse ::= SEQUENCE {
    certReqId            INTEGER,
    status               PKIStatusInfo,
    certifiedKeyPair     CertifiedKeyPair     OPTIONAL,
    rspInfo              OCTET STRING         OPTIONAL
    -- analogous to the id-regInfo-utf8Pairs string defined
    -- for regInfo in CertReqMsg [CRMF]
}

CertifiedKeyPair ::= SEQUENCE {
    certOrEncCert        CertOrEncCert,
    privateKey       [0] EncryptedValue       OPTIONAL,
    -- see [CRMF] for comment on encoding
    publicationInfo  [1] PKIPublicationInfo   OPTIONAL
}

CertOrEncCert ::= CHOICE {
    certificate      [0] Certificate,
    encryptedCert    [1] EncryptedValue
}
```

Código Fonte A.9: Estrutura CertRepMessage em ASN.1

```
RevReqContent ::= SEQUENCE OF RevDetails

RevDetails ::= SEQUENCE {
    certDetails          CertTemplate,
    crlEntryDetails      Extensions       OPTIONAL
}
```

Código Fonte A.10: Estrutura RevReqContent em ASN.1

```
RevRepContent ::= SEQUENCE {
    status         SEQUENCE SIZE (1..MAX) OF PKIStatusInfo,
    revCerts   [0] SEQUENCE SIZE (1..MAX) OF CertId OPTIONAL,
    crls       [1] SEQUENCE SIZE (1..MAX) OF CertificateList
                   OPTIONAL
}
```

```
CertId ::= SEQUENCE {
      issuer              GeneralName,
      serialNumber        INTEGER
}
```

Código Fonte A.11: Estrutura RevRepContent em ASN.1