PATRICIA DOUSSEAU CABRAL

IMPLEMENTAÇÃO DO PROTÓTIPO DE UMA INFRAESTRUTURA DE CHAVES PÚBLICAS BASEADA EM CERTIFICADOS AUTO-ASSINADOS

Florianópolis

2011

PATRICIA DOUSSEAU CABRAL

IMPLEMENTAÇÃO DO PROTÓTIPO DE UMA INFRAESTRUTURA DE CHAVES PÚBLICAS BASEADA EM CERTIFICADOS AUTO-ASSINADOS

Dissertação submetida ao Programa de graduação da Universidade Federal de Santa Catarina para a obtenção do Grau de Bacharel em Ciências da Computação.

Orientador: Cristian Thiago Moecke

Coorientador: Ricardo Felipe Custódio

UNIVERSIDADE FEDERAL DE SANTA CATARINA

Florianópolis

2011

Monografia de Conclusão de Curso apresentada para a obtenção de grau de Bacharel em Ciências da Computação pela Universidade Federal de Santa Catarina.

Implementação de uma Infraestrutura de Chaves Públicas baseada em Certificados Auto-Assinados

Autora: Patricia Dousseau Cabral

Orientador:

Cristian Thiago Moecke (Orientador) (UFSC)

Coorientador:

Prof. Dr. Ricardo Felipe Custódio (Coorientador) (UFSC)

Banca Avaliadora:

Jonathan Gehard Kohler (UFSC)

Lucas Martins (UFSC)

Marcelo Carlomagno Carlos (University of London)

Ao Vitor, por me ensinar a clareza da vista, e aos meus pais, por me darem a alma com que a ver clara.

AGRADECIMENTOS

Agradeço ao Professor Ricardo Felipe Custódio e aos colegas do laboratório por me permitirem descobrir o quão desafiador é segurança em computação.

Ao Cristian por ter me ajudado no desenvolvimento deste trabalho e por ter pacientemente respondido a todos os meus emails.

SUMÁRIO

Lista de Abreviaturas e Siglas

AC ..... *Autoridade Certificadora*
AV ..... *Autoridade de Validação*
AR ..... *Autoridade de Registro*
ACT ..... *Autoridade de Carimbo de Tempo*
ICP ..... *Infraestrutura de Chaves Públicas*
SPKI ..... *Simple Public Key Infrastructure*
PKI ..... *Public Key Infrastructure*
PGP ..... *Pretty Good Privacy*
ASN.1 ..... *Abstract Syntax Notation One*
SGBD ..... *Sistema de Gerenciamento de Banco de Dados*
DER ..... *Distinguished Encoding Rules*
UML ..... *Unified Modeling Language*
CGI ..... *Common Gateway Interface*
XML ..... *Extensible Markup Language*
HTML ..... *HyperText Markup Language*

LISTA DE FIGURAS

RESUMO

Com a rápida expansão da Internet, novas exigências relacionadas à segurança tornam-se vitais. Uma das importantes ferramentas de segurança em sistemas computacionais são as Infraestruturas de Chaves Públicas (ICP) que permitem assegurar a identidade do usuário ou sistema. Com isso, ela oferece uma mediação de credibilidade e confiança em comunicações entre as partes envolvidas em uma transação eletrônica. Com a popularização do uso de certificados digitais para os mais diversos usos, inclusive assinatura digital de documentos, algumas limitações da abordagem de Infraestrutura de Chaves Públicas (ICP) tradicionalmente utilizada tornaram-se ainda mais evidentes, como a dificuldade de implementação e de prestação de serviços à longo prazo. Diante da necessidade de um novo modelo de ICP mais eficiente, seguro e vantajoso, foi proposto em [1] uma nova abordagem de ICPs baseada em certificados digitais autoassinados e Autoridades de Validação. O presente trabalho mostra as vantagens e a viabilidade do novo modelo, validando-o e aprimorando-o através da implementação e utilização de um protótipo. O protótipo é composto por uma Autoridade de Registro e uma Autoridade de Validação. Além disso, foi implementado um analisador de tokens, para facilitar a interpretação das provas de validade emitidas pela Autoridade de Validação.

Palavras-chave: Criptografia, Infraestrutura de Chaves Públicas, ICP, Assinatura Digital

ABSTRACT

With the rapid expansion of the Internet, new security-related requirements became vital. An important security tool for computational systems is the Public Key Infraestructure (PKI) which ensures the user's or system's identity. It offers a mediation of credibility and trust in communications between parties in electronic transactions. With the popularization of the use of digital certificates for different purposes, including digital signature of documents, some limitations of the traditional approach to Public Key Infrastructure (PKI) became increasingly evident, such as the difficulty of long term implementation and provision of services. Facing the need for a safer, more efficient new ICP model, it was proposed in [1] a new approach based on self-signed digital certificates and Validation Authority. This paper shows the advantages and feasibility of the new model, validating and improving it through the implementation and use of a prototype. The prototype consists of a Registrarion Authority and a Validation Authority. In addition it was implemented a token analyzer, to help interpreting the validity evidences issued by the Validation Authority.

Keywords: Cryptography, Public Key Infrastruture, PKI, Digital Signature

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Com a popularização da Internet e, consequentemente, de transações online, surge uma necessidade crescente de assegurar a autenticidade de sites e de pessoas. Num ambiente onde não se pode mostrar credenciais físicas, é necessário um mecanismo que dê suporte a confidencialidade, integridade, autenticidade e não repúdio das informações [2].

A Infraestrutura de Chaves Públicas surgiu como uma alternativa para prover suporte aos quatro itens acima, sendo, hoje, um dos mecanismos de segurança mais utilizados para estes fins.

Sites de comércio eletrônico, assinatura de documentos digitais, transações automáticas, *home banking*, email, entre outros, todos utilizam certificação digital como meio para assegurar a segurança das informações [2]. Se, por um lado é, incontestável a importância das ICPs hoje, por outro, verificamos as inúmeras dificuldades em implementá-las e mantê-las. Além disso, existe um custo significativo envolvido no desenvolvimento e manutenção de uma ICP [3].

Portanto é necessário encontrar modelos mais viáveis em termos de eficiência e segurança. Moecke et al. [1] apresenta um modelo alternativo de ICP, onde buscam simplificar a implantação e utilização das ICPs e prover serviços mais adequados à longo prazo. Este trabalho tem por objetivo implementar e validar tal proposta, através da implementação e utilização de um protótipo.

1.2 OBJETIVOS

1.2.1 *OBJETIVO GERAL*

O objetivo geral deste trabalho é implementar o novo modelo de Infraestrutura de Chaves Públicas proposto em [1] e mostrar sua viabilidade em ser utilizado na prática e vantagens em comparação com o modelo tradicional.

1.2.2 *OBJETIVO ESPECÍFICOS*

- Descrever as diferenças entre o modelo atual e o modelo proposto
- Implementar uma Autoridade de Validação e uma Autoridade de Registro
- Mostrar a viabilidade do novo modelo e quais suas vantagens

- Sugerir correções e melhorias para a abordagem, baseado nos resultados da implantação do protótipo

1.3 JUSTIFICATIVA E MOTIVAÇÃO

O objetivo de uma Infraestrutura de Chaves Públicas é prover mecanismos que permitam aos sistemas implantar métodos de autenticação, identificação automática, funções de autorização e controle de acesso [4]. Os usuários de uma ICP podem ser desde pessoas até processos, sejam por ter um certificado emitido em seu nome ou por utilizar um serviço que faz uso de certificação digital. Com isso podemos ver a amplitude de uma Infraestrutura de Chaves Públicas e sua importância.

Por vários anos tem se tentado construir ICPs utilizando os métodos e mecanismos descritos nos padrões X.500 e também nas mais de duas mil páginas de especificações da IETF (Internet Engineering Task Force) [5]. Durante todo esse tempo a viabilidade e praticidade desta tecnologia foi posta em questão devido ao grande número de experiências negativas decorrentes da dificuldade de sua implementação [5].

Um novo modelo proposto por [1] busca resolver os problemas das ICPs tradicionais. Esta abordagem é recente e inédita, e para seu aprimoramento é importante a aplicação prática das idéias apresentadas através da implementação de um protótipo. Através deste protótipo, limitações, pontos fortes e características em geral da abordagem ficarão mais claros, permitindo seu aprimoramento.

1.4 METODOLOGIA

Primeiramente foi realizado um estudo aprofundado na área de Criptografia Assimétrica, principalmente sobre Infraestrutura de Chaves Públicas e sobre os diversos modelos de certificação digital existentes, através de livros, sites e artigos, em especial o de Moecke et al. [1].

Baseado nisto, foi proposta uma modelagem do software, através de diagramas de classe e de atividade, e verificou-se quais seriam as características fundamentais para o seu desenvolvimento. Tendo isso por base, foram escolhidas as ferramentas que seriam utilizadas através de pesquisas e comparações entre suas diversas funcionalidades e bibliotecas.

A cada nova etapa concluída no desenvolvimento do projeto, foram realizados testes para verificar a corretude do programa e para a correção de bugs e falhas de segurança.

Como resultado, foi criada uma Infraestrutura de Chaves Públicas parcial, composta por Autoridade de Registro e Autoridade de Validação, baseada no modelo [1], permitindo assim a verificação de suas vantagens e viabilidade.

1.5 ESTRUTURA DO DOCUMENTO

Este documento encontra-se estruturado da seguinte forma. O Capítulo 2 referente a *Fundamentação Teórica* contém uma visão geral sobre Criptografia Assimétrica e Infraestrutura de

Chaves Públicas, sua história, a necessidade de sua criação e suas principais utilizações.

No Capítulo 3, *Características e problemas de uma ICP comum*, é mostrado o estado da arte das ICPs. No Capítulo 4, é descrito o novo modelo proposto por Cristian et al. [1].

O Capítulo 5, Implementação, diz respeito a parte de projeto e desenvolvimento da ICP, quais ferramentas foram utilizadas e o porquê de sua escolha, modelagem proposta e as dificuldades encontradas.

E, por fim, no Capítulo 6, de Considerações Finais, são abordadas as vantagens do novo modelo, a justificativa de sua viabilidade e quais os trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 HISTÓRIA

Dois matemáticos de Stanford (Califórnia, Estados Unidos), Whitfield Diffie e Martin Hellman, publicaram, em 1976 no journal IEEE Transactions on Information Theory, um paper intitulado New Directions in Cryptography [6], comentando um dos grandes problemas da criptografia simétrica: o compartilhamento da chave de forma segura.

Em seu artigo, Diffie e Hellman não propunham a solução final para o problema, mas sim uma solução parcial para a questão da distribuição de chaves [7], que hoje é conhecida como protocolo de acordo de chaves Diffie-Hellman. Este protocolo deu base para o desenvolvimento da criptografia assimétrica, que permite que uma das chaves seja distribuída através de um canal de comunicação sem que seja necessário se preocupar se este canal é seguro ou não, pois essa chave se tornará pública, enquanto que a outra, que é utilizada para cifrar, é mantida em segredo.

Desde então, um grande número e variedade de criptografia, assinatura digital, acordo de chaves e outras técnicas tem sido desenvolvidas no campo da criptografia de chaves públicas.

2.2 ACORDO DE CHAVES DE DIFFIE-HELLMAN

O acordo de chaves de Diffie-Hellman é um método que permite que duas pessoas troquem chaves secretas entre si em um canal inseguro, sem nenhum segredo previamente estabelecido. Podemos ver através da Figura 1 abaixo que a primeira parte do protocolo exige que as duas partes concordem sobre dois números que serão usados nos cálculos da chave, sendo um número primo e outro uma raiz primitiva (número base). Em seguida, cada parte seleciona um número que será apenas de seu conhecimento (número secreto). Agora utilizando os dois números (o que foi acordado e o número particular de cada um) calculam um novo número primo que será novamente trocada entre os dois. Com isso, agora tem-se informação suficiente para calcular a chave privada compartilhada, que será gerada por cada uma das partes e será exatamente igual, sem a necessidade de transmiti-la pela rede [8].

Um fato importante é que esse protocolo é suscetível ao ataque do homem do meio. Uma terceira parte pode interceptar a comunicação e enviar seus valores para cada uma das partes, se passando por eles e com isto acordando uma chave com o primeiro e outra com o segundo.

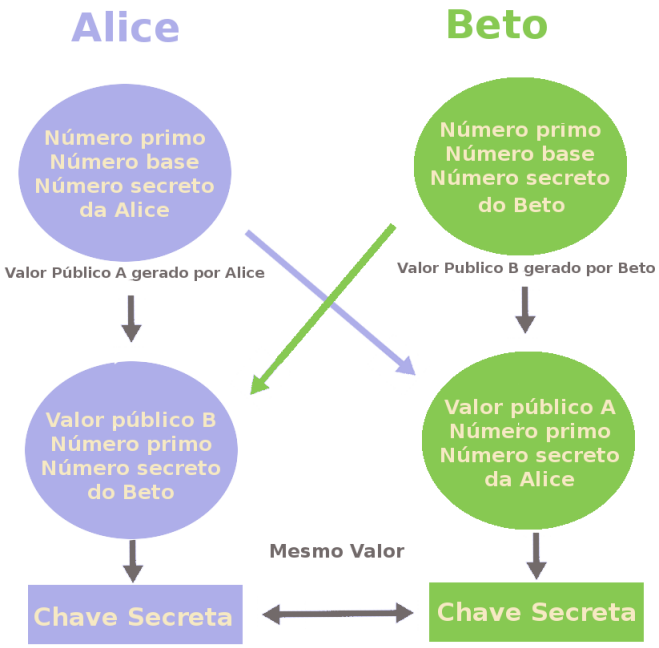

Figura 1: Visão geral do protocolo de Diffie-Hellman. Baseado em [9]

2.3 MÉTODOS CRIPTOGRÁFICOS

Existem duas técnicas de criptografia, a simétrica e a assimétrica. A simétrica é o modelo mais antigo e bem conhecido, onde a mesma chave cifra e decifra, ou seja, qualquer um que tenha acesso à chave terá acesso a qualquer mensagem que foi cifrada através dela.

Já a criptografia assimétrica é relativamente recente e faz uso de um par de chaves, que permitem cifrar e decifrar, e também assinar documentos digitais. Duas chaves são geradas ao mesmo tempo de forma que tenham uma relação única entre si, o que uma cifra apenas a outra decifra, e vice-versa. Escolhe-se uma das duas para ser a privada (que só o dono terá acesso) e a outra, pública, que será divulgada e se tornará acessível a quem desejar. Enquanto no modelo simétrico a chave deve ser de conhecimento privado, apenas entre as pessoas envolvidas, no modelo assimétrico uma das chaves pode ser livremente distribuída, eliminando assim o problema de como distribuir as chaves de forma segura [10].

2.4 CRIPTOGRAFIA SIMÉTRICA

O grande problema desse tipo de criptografia reside na distribuição da chave, pois nem sempre dispomos de canais realmente seguros para distribuí-la, além da dificuldade de armazenamento. Uma das vantagens deste tipo de criptografia é que ela normalmente é mais rápida que os algoritmos de criptografia assimétrica, sendo ambas comumente empregadas juntas, uma pelo quesito velocidade e outra pelo quesito segurança e facilidade.

Na Figura 6 temos um documento sendo cifrado com uma chave secreta e como resultado obtemos um documento cifrado. Para decifrá-lo utilizamos essa mesma chave e temos novamente o texto original.

Figura 2: Exemplo de criptografia simétrica

2.5 CRIPTOGRAFIA ASSIMÉTRICA

Como dito anteriormente, a criptografia assimétrica (também conhecida como criptografia de chaves públicas) é um método que utiliza duas chaves: uma pública, que será livremente distribuída, e outra privada, que será mantida em segredo.

Para exemplificar, tomemos como exemplo duas pessoas, Alice e Beto, que querem estabelecer uma comunicação. Beto gera um par de chaves, mantém uma em segredo (privada) e divulga a outra (pública). Alice, conhecendo a chave de Beto, pode agora utilizar essa mesma chave para cifrar documentos que serão decifrados apenas por Beto, pois ele é o único que detém a chave privada.

Figura 3: Exemplo de criptografia assimétrica utilizada para confidencialidade

Na Figura 3 temos um documento sendo cifrado com a chave pública de Beto. Como apenas o que uma cifra a outra decifra, apenas Beto será capaz de ler o conteúdo do documento cifrado, pois apenas ele tem acesso a chave privada. Com isso, temos confidencialidade, pois apenas o detentor da chave privada pode decifrar a mensagem.

2.6 ASSINATURA DE DOCUMENTOS ATRAVÉS DE CRIPTOGRAFIA ASSIMÉTRICA

Da mesma forma que Alice utilizou a chave pública de Beto para cifrar documentos, Beto pode utilizar a sua chave privada para cifrar documentos que serão decifrados por qualquer

pessoa que conheça a sua chave pública. Uma vez que a relação entre as chaves é única, sabemos que a única pessoa capaz de cifrar um documento decifrado com a chave pública de Beto é o próprio Beto. Sendo assim, podemos ter certeza de quem cifrou o documento, ou melhor, podemos usar isto como uma forma de autenticação, utilizando esse mecanismo para assinar documentos digitais. A pessoa que assinou não pode negar que foi ela quem realmente assinou, pois é a única que detém acesso à chave. A Figura 4 ilustra a assinatura de um documento usando criptografia assimétrica.

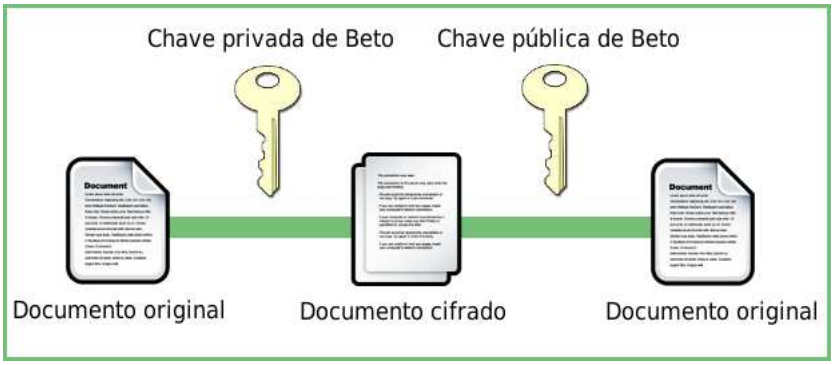

Figura 4: Exemplo de criptografia assimétrica usada para autenticação

Nesse caso, é gerado o resumo criptográfico (hash) do documento que se deseja assinar e ciframos esse resumo criptográfico com a chave privada de Beto, obtendo assim a assinatura do documento. Quando terceiros desejam validar essa assinatura, utilizam a chave pública de Beto verificando que foi realmente Beto quem assinou. E para verificar que o documento não foi modificado por terceiros, geram o resumo criptográfico do documento enviado e comparam com o resumo criptográfico assinado. Caso batam, significa que o documento assinado por Beto não foi alterado.

2.7 ASSINATURA DIGITAL

As assinaturas digitais servem para demonstrar a autenticidade de um documento ou mensagem digital, da mesma forma que assinaturas manuscritas. Além de serem mais difíceis de forjar que as assinaturas convencionais, garantem a integridade de um documento ou a autenticidade de e-mails.

Como cifrar documentos com o uso de criptografia assimétrica é muito menos eficiente em termos computacionais do que utilizando criptografia simétrica, normalmente é necessário gerar um resumo criptográfico (que é um identificador único) do arquivo e criptografá-lo com a chave privada, pois assim não é preciso criptografar o documento inteiro, o que levaria mais tempo.

2.8 CERTIFICADOS DIGITAIS

A técnica de criptografia assimétrica nos permite que documentos e mensagens sejam transmitidos e mantidos de forma segura. Mas com isso, surge a dificuldade de associar a chave com a pessoa, de forma que possamos ter garantia de que aquela chave é de determinado usuário, e

não de alguém se passando por ele. Para isto, foram criados os certificados digitais, que permitem que façamos essa conexão de forma segura. A segurança é garantida pela Autoridade Certificadora que é uma entidade que garante que aquela pessoa é realmente a detentora da chave e realmente é quem diz ser.

2.9 INFRAESTRUTURA DE CHAVES PÚBLICAS

Infraestrutura de Chaves Públicas permite que você tenha credibilidade nos certificados que foram assinados por determinada autoridade. Para isso precisamos de uma autoridade em que todas as partes possam confiar e que possa informar qual a chave pública de determinada entidade, através de certificados digitais. O atual modelo de ICP, bem como suas limitações, será explicado em maiores detalhes no próximo capítulo.

2.10 OUTROS MODELOS DE CERTIFICAÇÃO DIGITAL

2.10.1 *SPKI*

Simple Public Key Infrastructure (SPKI) é um padrão para Certificados de Chaves Pública com requisitos diferentes do propostos pela Infraestrutura de Chaves Públicas (ICP em português ou PKI em inglês). Sua motivação foi a dificuldade em gerenciar o controle de acessos a diferentes sistemas utilizando apenas o modelo ICP. Por exemplo, permitir que determinado usuário tenha permissão de escrita e leitura sobre certos arquivos pelas duas próximas semanas, enquanto outros usuários tenham apenas permissão de execução sobre os mesmo arquivos, mas apenas por dois dias. A SPKI propõe a solucionar problemas como este, permitindo a gerência de acesso a diferentes usuários, com diferentes permissões, sobre determinados serviços, possibilitando ou não delegar esses direitos [11].

O modelo SPKI propoe uma estrutura diferente de certificados digitais, sendo necessário especificar quem emitiu o certificado, quais permissões serão concedidas, a quem essas permissões serão concedidas, a validade do certificado e se existe a possibilidade de delegar esses direitos a outros usuários ou entidades. Outro conceito bastante diferente do usado pelas ICP tradicionais é a resolução de nomes. Enquanto nas ICPs é necessário fornecer um identificador único, o que é bastante difícil em um contexto global, as SPKI não exigem um identificador único e dizem respeito a um universo mais específico e reduzido, como uma empresa, funcionários ou uma determinada organização. Sendo assim, é possível, por exemplo, fornecer permissão de leitura para as irmãs de determinado usuário, mesmo que este possua mais de uma irmã, pois, como dito anteriormente, não é necessário um identificador único, mesmo dentro de um contexto específico. Isto é possível porque o modelo é dividido em duas partes: a cadeia de nomes, que permite chegar do emissor ao receptor, e a entidade que define o nome [11].

2.10.2 *PGP*

Pretty Good Privacy (PGP) é um software de criptografia criado em 1991 por Philip Zimmermman, disponibilizado de forma gratuita e utilizado principalmente em mensagens de e-mail eletrônico para aumentar sua segurança. O PGP é um modelo não hierarquizado onde os

próprios usuários validam uns aos outros. Cada usuário tem um conjunto de pessoas em quem conhece e confia. Os usuários trocam suas chaves públicas com outros usuários, aceitando as chaves uns dos outros e dos amigos dos outros usuários. Ou seja, segue-se o princípio de confiança onde se acredita na chave dos amigos de seus amigos. Assim é criada uma rede de credibilidade. Quando uma mensagem criptografada chega, é possível descriptografá-la se a chave estiver na rede de credibilidade do usuário. Temos assim uma rede de confiança [12].

3 CARACTERÍSTICAS E PROBLEMAS DE UMA ICP COMUM

Uma Infraestrutura de Chaves Públicas é um conjunto de tecnologias e procedimentos que provê segurança e confidencialidade no meio digital, emitindo chaves públicas, gerando e revogando certificado e distribuindo-os de forma que seja possível confiar no detentor da chave.

Usualmente as ICPs tem sido implementadas seguindo o padrão X.509 [4]. ICPs neste padrão são formadas por Autoridades Certificadoras (AC), Autoridades de Registro (AR) e Autoridades de Carimbo de Tempo (ACT) [10].

3.1 VISÃO GERAL DO MODELO EM VIGOR

No modelo vigente o usuário ou a entidade se identifica perante a Autoridade de Registro, que envia os dados e a chave pública do requisitante para a Autoridade Certificadora, que emite um certificado assinado com a chave privada da AC. Para verificar uma assinatura, precisamos analisar toda a cadeia de assinaturas até chegarmos a uma AC que confiamos. Além disso, é necessário uma Autoridade de Carimbo de Tempo para garantir que aquela assinatura foi gerada em um período que o certificado era válido.

3.2 AUTORIDADE CERTIFICADORA

A Autoridade Certificadora é a responsável por gerenciar o ciclo de vida do certificado, sendo encarregada de emitir, revogar e atualizar certificados digitais e gerar a Lista de Certificados Revogados (LCR). Por ter um papel fundamental e para diminuir a sobrecarga, alguns modelos adotam uma hierarquia entre as Autoridade Certificadoras, tendo como autoridade máxima a AC Raiz, e como subordinados, outras Autoridades Certificadores, em diferentes níveis. A Figura 5 demonstra a estrutura de uma ICP típica.

Uma das funções da Autoridade Certificadora é emitir certificados com a chave pública do usuário, de forma que seja possível associar de forma segura e confiável o dono à chave. Para isto, ela emite um certificado com os dados do detentor da chave (nome, e-mail, validade do certificado, organização, etc.) e o assina com a chave da própria Autoridade Certificadora. Uma vez que confiamos nessa AC, podemos confiar no certificado assinado por ela.

Porém é necessário um mecanismo que permita à AC acreditar que aquela chave realmente é de determinada pessoa ou entidade, responsabilidade esta que pode ser passada para a Autoridade de Registro.

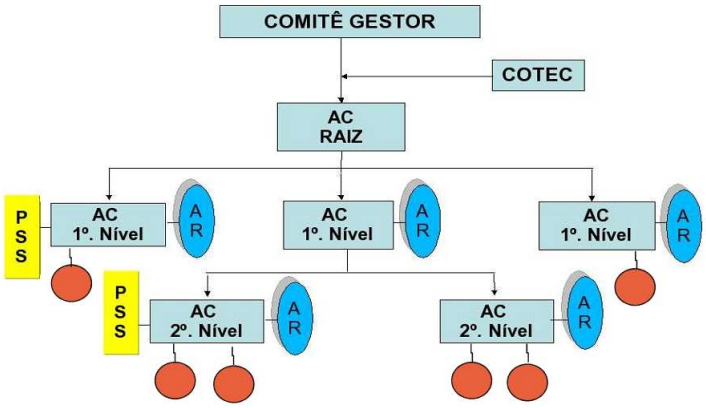

Figura 5: Exemplo de hierarquia da Autoridade Certificadora da ICP-Brasil

3.3 AUTORIDADE DE REGISTRO

É uma interface entre a Autoridade Certificadora e o usuário cuja principal função é verificar e assegurar quem é o verdadeiro dono de determinada chave. Ela é responsável por conferir as informações do usuário e enviar a requisição do certificado para a AC, com os dados do requisitante junto com sua chave pública. A AC irá emitir o certificado com base nos dados enviados pela Autoridade de Registro (AR).

3.4 AUTORIDADE DE CARIMBO DE TEMPO

Certificados Digitais possuem uma data de validade inicial e final que pode variar entre segundos até anos, por medidas de segurança e porque o certificado das AC e das AR tem uma data limite, impossibilitando que o certificado do usuário dure para sempre. Sendo assim, certificados expiram e podem ser revogados pelo usuário (um certificado pode ser revogado, por exemplo, caso a chave privada seja comprometida).

Não é possível confiar na data informada na própria assinatura quando o certificado foi revogado ou expirou, pois essa assinatura (incluíndo a data) pode ter sido forjada [14]. Diante desta situação, é necessário um mecanismo que nos permita confiar que uma determinada assinatura foi gerada em um período em que o certificado ainda era válido. Para isso, existem as Autoridades de Carimbo do Tempo (ACT) [15], que permitem ancorar de forma confiável a assinatura a uma data segura.

Um carimbo de tempo é similar à uma assinatura digital, com a diferença que além dos dados normalmente presentes, contém um data gerada por um relógio seguro. Esta data, sendo fornecida pela Autoridade de Carimbo do Tempo, pode ser considerada confiável. Carimbos de tempo também possuem uma data de expiração, pelos mesmos motivos que um certificado digital, exigindo, assim, que seja necessário criar novos carimbos de tempo em cima dos antigos, de forma que possamos sempre verificar que aquela assinatura foi gerada num período em que seu certificado ainda era válido.

3.5 CAMINHO DE VALIDAÇÃO

Como o certificado pode ser assinado por diversas autoridades, uma dando credibilidade a outra, temos com isso uma cadeia de certificação. Para verificar se determinada assinatura é válida, precisamos verificar o carimbo de tempo e o certificado da Autoridade de Carimbo de Tempo. Para verificar o certificado da ACT, verificamos o certificado da Autoridade Certificadora que assinou o seu certificado. Mas também precisamos verificar o certificado dessa AC, para saber quem o assinou até chegarmos a alguma autoridade que confiamos. Tudo isso apenas para verificar se o carimbo de tempo é válido. O mesmo é necessário para verificar a validade da assinatura: precisamos verificar todas as AC da cadeia de certificação até chegarmos na AC-Raiz ou em uma AC que confiamos.

3.6 LISTA DE CERTIFICADOS REVOGADOS

Para verificarmos se o certificado foi revogado, precisamos verificar a Lista de Certificados Revogados (LCR), que contém uma entrada para todos os certificados que foram revogados pelo detentor da chave. A LCR é emitida e assinada pela Autoridade Certificadora em intervalos de tempo predefinidos e disponibilizada na rede de forma que seja acessível a qualquer um que desejar conferir a validade de um certificado digital [13].

3.7 LIMITAÇÕES DO MODELO EM VIGOR

O modelo em vigor apresenta algumas limitações que dificultam o uso e a manutenção de assinaturas e certificados.

3.7.1 *LISTA DE CERTIFICADOS REVOGADOS*

Existem diversos problemas relacionados a existência da Lista de Certificados Revogados (LCR). Ela viola o princío de consistência dos dados, gerando um comportamento não determinístico: "o certificado é válido até sua data de expiração, ou até que digam outra coisa" [16].

Outro sério problema é em relação ao intervalo de emissão da LCR. Cada tipo de aplicação tem exigências diferentes em relação a esse intervalo, sendo este intervalo um fator crítico ou não. Por exemplo, em um sistema de cartões de créditos a criticidade é maior, devendo ser recuperada com maior frequência a LCR do que, por exemplo, um sistema de envio de emails.

Então como podemos ter um intervalo comum para diferentes aplicações? Para isso, poderíamos simplesmente emitir novas LCR com um intervalo de tempo pequeno e esperar que cada aplicação recuperasse esta lista, dependendo da criticidade da aplicação. Algumas recuperariam em intervalos maiores e outras em intervalos menores, mas quanto maior a frenquência de emissão, maior a sobrecarga sobre o servidor mantenedor das LCR. Ainda mais se considerarmos que a grande parte das aplicações usam apenas algumas entradas de uma lista que pode ser consideravelmente grande. Isso pode se tornar algo relativamente custoso, dependendo da frequência de emissão de novas LCR.

A emissão de LCR é a operação mais custosa para uma Autoridade Certificadora. Os usuários normalmente pagam uma taxa para poderem emitir um certificado, mas é tido como obrigação da AC emitir a LCR de forma gratuita, o que pode ser algo bastante desmotivante, uma vez que custa tempo de processamento e recursos físicos [16].

3.7.2 *CADEIA DE CERTIFICAÇÃO*

Para validar um certificado é necessário construir uma cadeia de certificação entre o certificado folha (o que se quer validar) até um AC de nível superior em que se tenha confiança. Analisar um simples certificado se transforma em uma operação extremamente custosa, uma vez que devemos verificar inúmeros certificados, dependendo do tamanho da hierarquia entre as AC. Caso exista certificação cruzada, o problema pode se tornar ainda mais complexo, porque teremos ciclos e, em casos mais extremos, a semântica do certificado muda conforme a montagem da cadeia [16].

3.7.3 *CARIMBO DE TEMPO*

Outro ponto negativo é que devemos sempre adicionar novos carimbos de tempo, assim que estes estejam para expirar, sob risco de perder definitivamente a validade do documento. Com isso acumulamos sempre novos carimbos, aumentando significantemente o tamanho da assinatura dos documentos [14]. Além disso, temos que sempre ter o cuidado de adicionar novos carimbos antes que os anteriores expirem, senão perdemos a validade da assinatura do documento.

4 ICP BASEADA EM CERTIFICADOS DIGITAIS AUTOASSINADOS

O novo modelo de Infraestrutura de Chaves Públicas proposto por Moecke et al. [1] prevê algumas modificações. É eliminado do modelo a Autoridade de Carimbo de Tempo, os certificados passam a ser assinados pelo detentor da chave e não mais pela AC. Uma nova entidade é criada, a Autoridade de Validação, responsável por emitir provas indicando a situação dos certificados da ICP.

4.1 MODELO PROPOSTO

No modelo tradicional, quando um certificado é emitido, ele é assinado por uma Autoridade Certificadora, e é válido até que expire ou seja revogado. Como o certificado pode ser revogado a qualquer momento, não sabemos se ele é válido a menos que consultemos a lista de certificados revogados e verifiquemos a assinatura da AC. Ou seja, assumimos que o certificado é sempre válido (uma vez que foi assinado por uma AC) até que se prove o contrário (que ele esteja na lista de certificados revogados). Mas poderíamos supor o inverso: que ele nunca é válido, até que se tenha uma prova de sua validade. E é exatamente isto que o novo modelo propõe: que o certificado seja autoassinado pelo usuário e que seja considerado válido apenas mediante uma prova de sua validade.

Retirando as Autoridades Certificadoras, não temos mais um caminho de certificação, pois não é mais necessário verificar todas as assinaturas até chegar na AC-Raiz ou a alguma AC Confiável. Grande parte dos problemas relacionados a isso são automaticamente eliminados [1].

A prova de validade é emitida por uma nova autoridade, a Autoridade de Validação, que tem por objetivo emitir *tokens*, que são as provas que comprovam a validade de determinado certificado.

4.1.1 *AUTORIDADE DE VALIDAÇÃO*

A Autoridade de Validação é a responsável por emitir provas de que determinado certificado é válido. Para isto, ela emite *tokens* de pequena duração, que expiram logo, para não ser necessário revogá-los. Caso o *token* fosse de longa duração, seria possível que o certificado expirasse ou fosse revogado antes do *token* perder sua validade, o que criaria uma inconsitência e a necessidade de revogá-lo.

Quando uma entidade deseja verificar a validade de determinado certificado, ela envia o identificador do certificado para a AV, como, por exemplo, seu resumo criptográfico, para que

seja verificado se este certificado foi ou não revogado, e recebe como resposta um *token* com todas as informações necessárias atestando a validade do certificado.

4.1.2 *AUTORIDADE DE REGISTRO*

A Autoridade de Registro continua com um papel fundamental dentro do modelo. Uma vez que o certificado é autoassinado, ele pode ser emitido a qualquer hora pelo usuário, que só precisa se dirigir a uma AR quando deseja que seu certificado seja validado por terceiros. Para isto, o usuário deve se dirigir presencialmente até uma AR e proceder como no modelo vigente, mostrando documentos que comprovem que ele realmente é quem alega ser.

Assim, a AR verifica os dados do certificado e a posse da chave privada e os envia para a Autoridade de Validação, através de uma mensagem que vincula o certificado ao usuário [1].

4.1.3 *AUTORIDADE CERTIFICADORA*

Apesar dos certificados não serem mais assinados pela Autoridade Certificadora, ela continua presente no modelo. Ainda existe uma Autoridade Certificadora Raiz que é responsável por assinar o certificado das Autoridades de Registro e das Autoridades de Validação, ou seja, ainda existe um caminho de certificação, mas muito menor, e que precisa ser verificado apenas uma vez para cada Autoridade, e não mais para cada certificado. Ainda há um lista de certificados revogados, mas sensivelmente menor e consequentemente mais fácil de ser gerenciada.

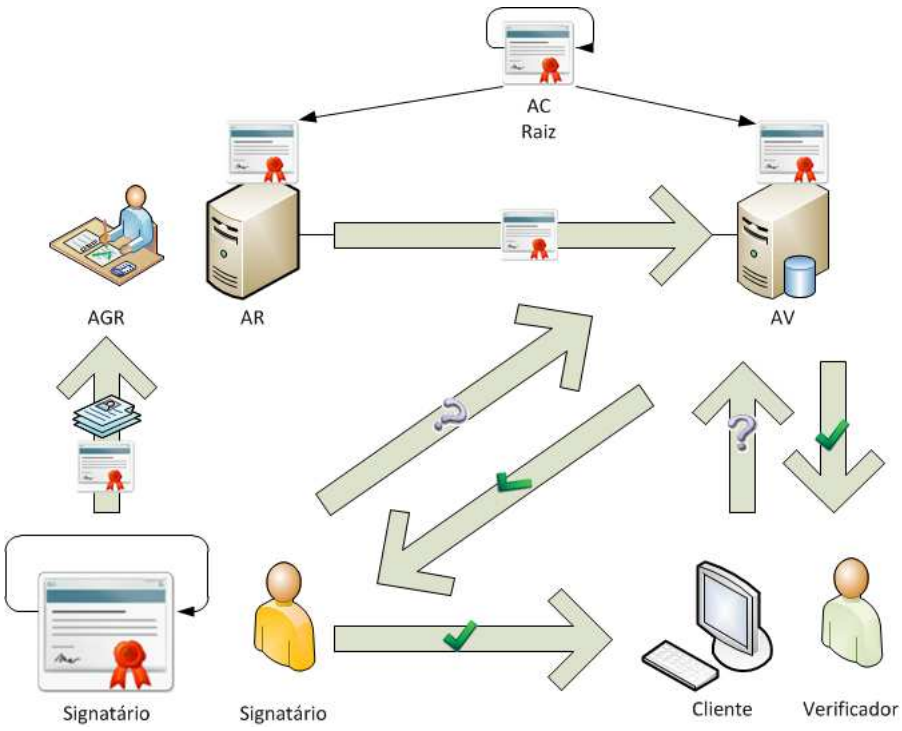

Figura 6: Arquitetura de ICP proposta [1]

4.2 TOKEN

Os certificados passam a ser emitidos pelo usuário e não são mais assinados pela Autoridade Certificadora. Mas é necessário uma prova para comprovar a sua validade. Essa prova é o *token*, que é emitido pela AV sempre que uma entidade deseja verificar um certificado ou uma assinatura.

Tokens têm data de validade, expirando depois de determinado tempo. Têm preferencialmente uma curta duração, para que não seja necessário revogá-los e crir uma lista de *tokens* revogados, trazendo todos os problemas inerentes a essas lista, como já discutido anteriormente. Mesmo depois que um token expira, ele pode continuar sendo utilizado para provar que aquele certificado ou aquela assinatura eram válidos no período de tempo em que o token era válido, já que ele tem uma data de início e fim de validade.

Tokens contêm toda informação necessária sobre a validade de um certificado ou assinatura. Através deles é possível saber a qual certificado fazem referência, se o certificado é válido, e caso não seja, o motivo de não ser. Quando foi emitido e quando expira, o tipo de algoritmo utilizado para gerar o resumo criptográfico do certificado, entre outras informações relevantes.

Todo *token* é assinado pela Autoridade Validadora, para termos certeza que foi ela quem o emitiu. Assim não é possível que seu conteúdo seja alterado ou forjado por outrém, sendo confiável [1].

4.3 DIFERENTES USOS DE UM CERTIFICADO DIGITAL

Certificados digitais podem ser usadas tanto para autenticação perante um sistema ou alguma outra entidade, quanto para assinar documentos digitalmente. No primeiro caso, poderíamos tomar, por exemplo, um sistema que exija que o usuário se autentique através do seu certificado digital, comprovando que ele é realmente quem diz ser. No segundo, teríamos um usuário que usa de sua chave privada para assinar um documento, tornando inegável o fato de ter sido ele quem o assinou.

No modelo proposto de ICP, quando usamos um certificado para autenticação, devemos gerar um *token* de curta duração, mas que pode ser reaproveitado pelo sistema enquanto ele não expirar, evitando, assim, que o serviço seja obrigado a verificar a validade do certificado toda vez que o usuário se autenticar perante o sistema.

Já no caso de documentos assinados digitalmente, poderíamos ter um *token* válido apenas por um determinado instante de tempo, que comprovasse que o certificado era válido no instante em que o *token* foi emitido. Entretanto, para isso, é necessário que o *token* possua uma referência ligando-o à assinatura digital, para que se comprove que a assinatura existia na data em que o *token* foi solicitado: por exemplo, o resumo criptográfico da assinatura realizada. Esta abordagem é similar à de carimbos de tempo [18]. Poderíamos, então, anexar esse token à assinatura, possibilitando que outras entidades que desejam verificar a assinatura precisem apenas verificar o token, não precisando emitir uma nova prova da validade do documento.

4.4 VANTAGENS DO NOVO MODELO PROPOSTO SOBRE O ATUAL

O modelo proposto apresenta uma série de vantagens sobre o modelo tradicional de ICP baseado no padrão X.509. Entre elas, poderíamos citar:

- Eficiência e maior facilidade na verificação da validade de certificados, uma vez que não é necessário validar toda uma cadeia de certificação, mas apenas verificar a validade do token e de seu emissor;
- Possibilidade de se reutilizar o mesmo token várias vezes até sua data de expiração, evitando assim a emissão contínua de novas provas de validade, seja por quem o emitiu ou por terceiro;
- Simplificação do processo de validação e conservação de uma assinatura digital e dos dados do signatário;
- Maior flexibilidade na organização da ICP;

5 IMPLEMENTAÇÃO

5.1 FERRAMENTAS UTILIZADAS

5.1.1 *PYTHON 2.6.5*

Para implementar a Autoridade de Validação (VA) foi utilizada a linguagem de programação Python [19], por ser uma linguagem de rápida programação cuja sintaxe é clara e simples, e cujo código gerado é de fácil compreensão e normalmente menos extenso do que seria caso fosse feito em C/C++ ou Java. Outro ponto importante foi ter suporte para uma grande diversidade de bibliotecas externas, tais como M2Crypto (para criptografia) e MySQL-Python (para o SGBD MySQL). Uma característica fundamental é a integração com C/C++ caso fosse necessário utilizar bibliotecas escritas nestas linguagens ou escrever código de baixo nível. Outro ponto importante é a facilidade em criar mecanismos cliente-servidor ou fazer uma interface web. E por ser uma linguagem cuja comunidade é bastante ativa, sendo fácil achar documentação e exemplos sobre o que for necessário.

5.1.2 *M2CRYPTO 0.20.2*

Existem diversas bibliotecas criptográficas para Python. A M2Crypto foi escolhida por ser o mais completo wrapper para OpenSSL e ter várias funcionalidades essencias para a execução do projeto [20]. Podemos ver na Figura 7 que ela contém os tipos de certificados e codificações utilizados no trabalho (X509, ASN1 e PEM), a cifragem assimétrica RSA e uma ampla gama de hashes. Além de ser o único que roda em Linux, Windows e Mac OS X. Esses fatores foram decisivos para escolher o M2Crypto como biblioteca para o desenvolvimento do projeto.

5.1.3 *MYSQL 5.1.41*

Um sistema gerenciador de banco de dados (SGBD), como o próprio nome diz, tem por função gerenciar um banco de dados. Ele permite uma interação mais simples do usuário com a base de dados, através de comandos SQL para inclusão, alteração e remoção das informações armazenadas. Além disso, é responsável por controlar o acesso aos dados, sua manipulação e organização.

Foi necessário utilizar um SGBD para manipular os certificados cadastrados e todas as informações referentes a eles, como validade e situação do certificado.

O SGBD utilizado foi o MySQL [22], também por ser gratuito, robusto e com bastante documentação a respeito.

2010 (C) ActiveState

Comparison of Python cryptography modules

	M2Crypto	pycryptopp	cryptlib_py	python-nss	pycrypto	cryptopy
Version	0.20.2	0.5.17	3.2.2	0.8	2.0.1	1.2.5
Date	October 7, 2009	September 23, 2009	September 13, 2005	September 21, 2009	June 14, 2005	March 3, 2003
License	BSD-style	GPL	GPL-like[1], Commercial[2]	MPLv1.1[3]	Public Domain	Artistic License
Export Laws	USA	USA	New Zealand	USA	USA	USA
Type	Wrapper	Wrapper	Wrapper	Wrapper	C module	Pure Python
Underlying system	OpenSSL	Crypto++[4]	Cryptlib	NSS	N/A	N/A
Platform	CPython[5]	CPython	CPython	Cpython on Linux	CPython	Any
Python	<= 2.6	<= 2.5	2.x[6]	= 2.6	<= 2.6	2.2+
Binaries	Linux, Windows, Mac OS X	Linux	Windows	Fedora Linux/RHEL	Linux	N/A
Higher level API	Some	No	Full	Some	No	Some
Extensions	Loadable "engines" (shared libraries)	Not supported	Statically compiled C modules	Not supported	Not supported	Python modules
Technical details						
	M2Crypto	**pycryptopp**	**cryptlib_py**	**python-nss**	**pycrypto**	**cryptopy**
Symmetric Ciphers	Blowfish, CAST, CAST5, DES, Triple DES, IDEA, RC2, RC4, RC5, AES, Camellia, SEED	AES (Rijndael)	AES, Blowfish, CAST-128, DES, Triple DES, IDEA, RC2, RC4, RC5 Skipjack	AES, DES, Triple DES, RC4, RC2	AES, RC2, RC4, RC5, Blowfish, CAST, DES, Triple DES, IDEA, XOR	Rijndael, AES, RC4, WEP, TKIP
Asymmetric	DSA, RSA, Elliptic Curve DSA, GOST R 34.10 1994/2001	RSA	DSA, Elgamal, RSA	RSA, DSA, Elliptic Curve DSA	DSA, Elgamal, RSA	-
Key Agreement	Diffie-Hellman, Elliptic Curve Diffie-Hellman (ECDH)	-	Diffie-Hellman	RSA, Diffie-Hellman, Elliptic Curve Diffie-Hellman (ECDH)	-	-
Hashes	MD2, MD4, MD5, MDC2, RIPEMD-160, SHA-1, SHA-224/256/384/512	SHA-256	MD2, MD4, MD5, RIPEMD-160, SHA-1, SHA-2/SHA-256	SHA-1/SHA-256/SHA-384/SHA-512, MD5, MD2	MD2, MD4, MD5, RIPEMD, SHA, SHA-256	MD5, SHA-1
Message Authentication	HMAC, GOST 28147-89 MAC	-	HMAC-MD5, HMAC-SHA, HMAC-RIPEMD-160	HMAC	HMAC	HMAC, IEEE
RNG	Custom[7] algorithm based on SHA-1	ANSI X9.17 appendix C, RandomPool	/dev/random, EGD, PRNGD, ANSI X9.17/X9.31, hardware generators	FIPS 186-2	Randpool	Rijndael_256k_256b
Certificates, I/O and Encoding	X509, X509v3, ASN.1, PEM, S/MIME, PKCS#7, PKCS#12	-	X.509v1 to X.509v4, SET, S/MIME, PGP/OpenPGP, AuthentiCode, Identrus, SigG, Qualified Certificates, IPSec, PKCS#7, PKCS#11, PKCS#15	PKCS#5, #3, #5, #7, #8, #9, #10, #11, #12, S/MIME, X.509v3, OCSP	RFC1751	-
Supported Hardware	Through custom "engines"	-	PKCS#11 compatible, other	PKCS#11 compatible	-	-
Certification						
	M2Crypto	**pycryptopp**	**cryptlib_py**	**python-nss**	**pycrypto**	**cryptopy**
FIPS-140	Level 2[8]	Level 2	Yes[9]	Level 2	No	No
ITSEC/Common Criteria	No	No	Yes	No	No	No
NSA Suite B	No	No	No	No	No	No

Figura 7: Tabela comparativa de bibliotecas [21]

5.1.4 *MYSQL-PYTHON 1.2.3*

Biblioteca para a integração do código em Python com o SGBD MySQL, escolhida por ser bastante robusta e já bem estabelecida e fácil de usar [23].

5.1.5 *PYASN1 0.0.11*

Como os tokens seguem a codificação Abstract Syntax Notation One (ASN1), foi necessário utilizar uma biblioteca que permitisse descrever a estrutura dos tokens em ASN1. Para isto, foi escolhido o PyASN1 [24] por ter todas as funcionalidades necessárias para a criação dos tokens, como previsto em [1].

O padrão ASN.1 adotado foi o DER (Distinguished Encoding Rules).

5.1.6 *APACHE 2.2.14*

O sistema criado faz uso de páginas web para a comunicação com o usuário e scripts CGI, facilitando a sua interação com o sistema. Para isto, foi necessário utilizar um servidor web (Apache), que foi escolhido por ser software livre, gratuito, robusto e com bastante documentação a respeito [25].

5.1.7 *KOMODO EDIT 5.2.4*

Como Ambiente de Desenvolvimento foi utilizado o Komodo Edit [26], por ter sido o único aplicativo encontrado que permite uma fácil visualização da identação do código através

de marcações visuais, o que ajuda bastante em uma linguagem como Python. Outro ponto decisivo foi por ser gratuito.

5.1.8 *DBDESIGNER 4*

Para a modelagem do banco de dados foi utilizado o DBDesigner [27]. A modelagem foi feita para facilitar o projeto do banco de dados e também como uma forma de documentar o sistema.

5.1.9 *ASTAH COMMUNITY 6.2*

Para a modelagem UML (Unified Modeling Language) foi utilizado o aplicativo Astah Community [28], a versão gratuita do Astah. Foi necessário modelar o sistema para projetá-lo adequadamente. Isto facilitou bastante o seu desenvolvimento por ajudar a ter uma visão global do sistema. Também serviu como documentação.

5.2 INTERFACE COM O USUÁRIO

Para facilitar a comunicação do usuário com a Autoridade de Validação e a Autoridade de Registro, foram criadas duas páginas web, uma para interação com a Autoridade de Validação e outra para interação com a Autoridade de Registro. Também foi criada uma interface web para o Validador de Tokens.

5.2.1 *INTERFACE WEB DA AUTORIDADE DE VALIDAÇÃO*

A comunicação com a AV ocorre através do envio de um token de requisição e o recebimento de um token de resposta. Para facilitar a comunicação, foi criada uma página onde o usuário envia os dados necessários e um token é gerado automaticamente e enviado para a AV, que responde com outro token. O usuário tem então a possibilidade de fazer o download desta resposta para poder analisá-la ou enviá-la para alguma entidade.

Na Figura 8 vemos a página da AV, onde temos duas opções: verificar a validade de um certificado e verificar a validade da assinatura de um documento. No primeiro caso, é necessário enviar o hash do certificado que se deseja analisar. No segundo, é necessário entrar com a assinatura do documento a ser analisado, o hash do documento e o certificado que gerou aquela assinatura. O procedimento então segue igual para os dois casos, o usuário envia os dados requeridos e é gerado um token de resposta, que o usuário tem a opção de baixar e enviar para outro sistema que fará a validação do token e mostrará seu conteúdo em um formato mais legível ao usuário.

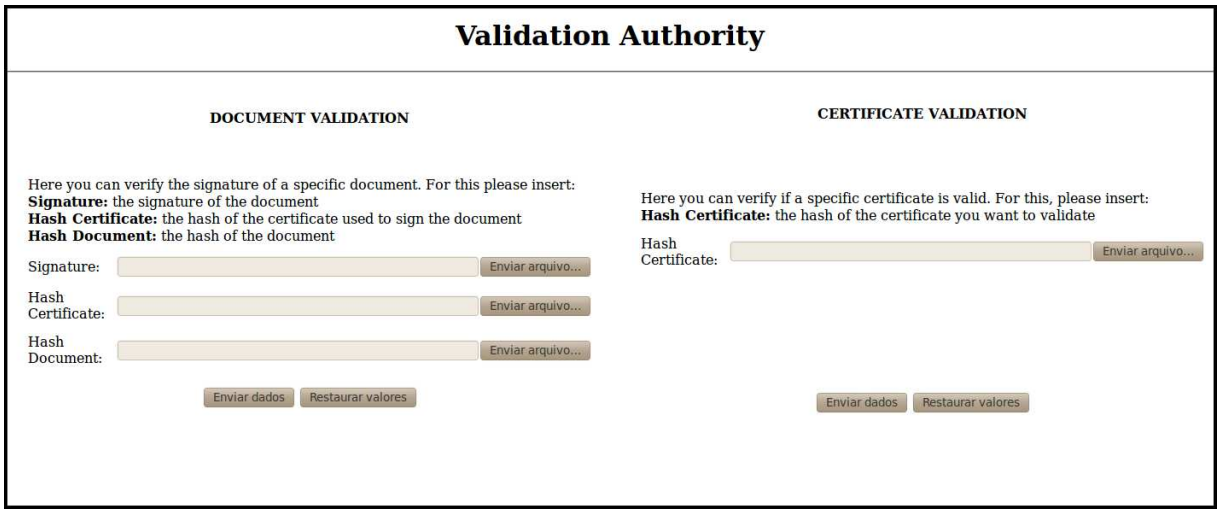

Figura 8: Página para interação com a AV

5.2.2 *INTERFACE WEB DA AUTORIDADE DE REGISTRO*

Diferentemente da comunicação com a Autoridade de Validação, a comunicação com a Autoridade de Registro não é feita através do uso de tokens, apenas nos conectamos ao servidor da AR e enviamos um pedido. A Figura 9 mostra a página da Autoridade de Registro, onde temos duas opções, podemos revogar um certificado já registrado ou cadastrar um novo certificado. Em ambos os casos deve-se enviar o hash do certificado e uma resposta aparecerá na tela, dizendo se a operação foi concluída com sucesso ou qual o erro ocorrido.

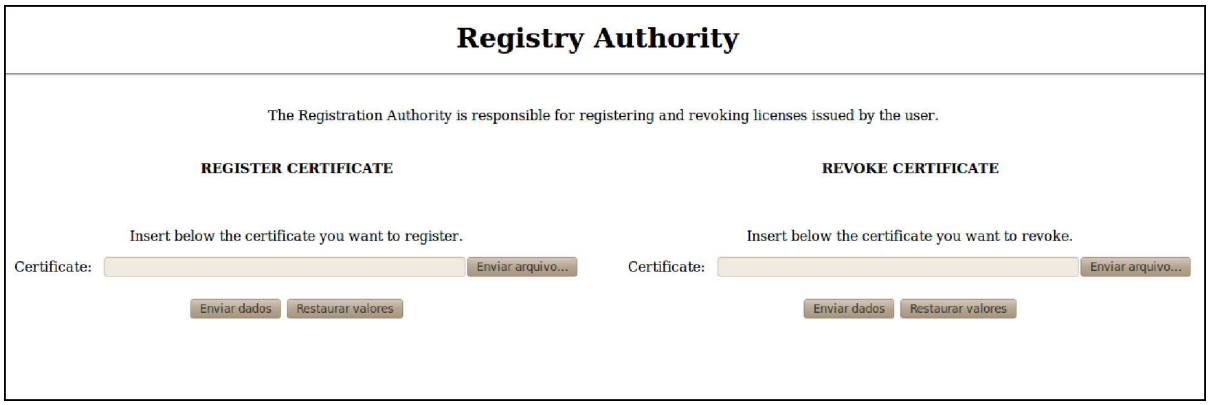

Figura 9: Página para interação com a AR

5.3 TOKEN

Tokens são emitidos pela Autoridade de Validação sempre que uma requisição é submetida a ela. Como tokens são codificados em ASN1, torna-se difícil ao usuário ler seu conteúdo. Foi então criado um mecanismo que facilita essa leitura e verifica se o token é válido, ou seja, se foi realmente assinado pela Autoridade de Validação. Os token são codificados em DER e tem uma estrutura definida de acordo com o que foi proposto em [1]. Eles foram implementados utilizando a biblioteca PyASN1, que gera o token em asn1 e o codifica em DER.

5.4 AUTORIDADE DE REGISTRO

A Autoridade de Registro é responsável por gerenciar certificados. Na modelagem proposta por Moecke et al. [1], certificados são emitidos pelo próprio usuário e enviados à Autoridade de Registro para cadastrá-los ou revogá-los. Certificados não contém mais a assinatura da Autoridade Certificadora, sendo necessário à AR manter um banco de dados com informações de todos os certificados cadastrados em seu sistema.

Toda vez que é necessário cadastrar ou revogar algum certificado, a Autoridade de Registro deve informar à Autoridade de Validação que determinado certificado foi revogado ou cadastrado, para que ela proceda da mesma forma no seu banco de dados. As informações da AV devem ser consistentes com as da AR. A comunicação entre as duas autoridades ocorre através do uso de arquivos xml assinados, contendo o tipo de requisição e os dados do certificado. Abaixo temos a estrutura do xml:

```
1  <?xml version="1.0" ?>
2  <message id="">
3     <registration_authority> hash do certificado da AR </
          registration_authority>
4     <request_type> REVOKE/REGISTER  </request_type>
5
6     <certificate_info>
7        <certificate>pem do certificado </certificate>
8        <digest>hash do certificado </digest>
9        <digest_algorithm> algoritmo utilizado para gerar o hash do
             certificado </digest_algorithm>
10    </certificate_info>
11
12    <signature>assinatura da AR</signature>
13 </message>
```

Podemos colocar quantos "certificate_info" forem necessários. Assim podemos revogar ou registrar vários certificados em uma única transação. Quando desejamos registrar um certificado, é necessário preencher o campo "certificate" podendo deixar o campo "digest" e "digest_algorithm" vazios. Precisamos enviar o certificado em PEM pois ele será mais tarde necessário para validar as assinaturas. Quando desejamos revogar um certificado, precisamos preencher os campos "digest" e "digest_algorithm" pois o certificado será buscado pela seu identificador para ser revogado. Nesse caso não é necessário seu PEM.

Para desenvolver a Autoridade de Registro, foram implementadas as seguintes classes:

- RegistrationAuthority
- BDMySQL
- SignedMessage
- RequisitionThread
- Configurations

A classe *RegistrationAuthority* possui toda a lógica da Autoridade de Registro, ela é responsável por cadastrar e revogar certificados, além de fazer a comunicação com a Autoridade

de Validação. Todas as operações realizadas são anotadas em um arquivo de log por medidas de segurança, caso seja necessário fazer auditoria do sistema. Caso não seja possível conectar-se a uma das AV que ela tenha contato, uma thread (*RequisitionThread*) é lançada em paralelo para continuar tentando se comunicar com a AV que não se obteve resposta, até que se atinja um número limite de tentativas. Atingido esse limite, a operação é anotada no log para consulta posterior.

Em [1] a comunicação entre a AR e a AV não estava especificada. Para isto, foi utilizado um xml assinado pela facilidade de utilizar e implementar este tipo de arquivo.

A classe *BDMySQL* faz o relacionamento com o banco de dados, permitindo revogar e cadastrar certificados, além de retornar o endereço de todas as Autoridades de Validação que a AR tem acesso. A modelagem dessa classe segue os mesmo princípios que os adotados no projeto da Autoridade de Validação, ser uma classe que pode ser facilmente desacoplada do resto do sistema, caso seja necessário trocar de SGBD. O SGBD utilizado foi o MySQL, por isto o nome da classe BDMySQL.

A classe *SignedMessage* é responsável por montar um xml autoassinado contendo o tipo de requisição (se é revogação ou registro de certificado) que será enviado para a Autoridade de Validação. Esse arquivo xml será assinado com a chave privada da Autoridade de Registro por medidas de segurança, para que ele não seja alterado ou forjado.

A classe *RequisitionThread* é uma thread que é lançada toda vez que não se obtém resposta de algum servidor responsável por fazer a comunicação com determinada Autoridade de Validação. Essa thread fica rodando em paralelo tentando reenviar o pedido para os servidores que não foram alcançados, até que se atinga um número limite de tentativas. Atingido esse limite, a operação é anotada no log para consulta posterior.

O arquivo *Configurations* contém todos os dados de configuração usados no sistema, como, por exemplo, o número máximo de tentativas para realizar determinada operação pela RequisitionThread. Este arquivo foi criado para evitar que dados de configuração sejam inseridos diretamente no código, dificultando qualquer alteração no sistema e possibilitando que as configurações fossem mais dinâmicas.

Na Figura 11 temos o diagrama de sequência para demonstrar os passos necessários para revogar um certificado. Primeiro a AR atualiza sua base de dados informando que aquele certificado foi revogado. Em seguida, manda uma mensagem para todas as AVs que mantém contato, pedindo que também revoguem este certificado. O diagrama de envio de mensagem foi suprimido por ser apenas uma conexão através de sockets com o servidor de cada uma das AV e o envio da mensagem criada anteriormente (xml assinado).

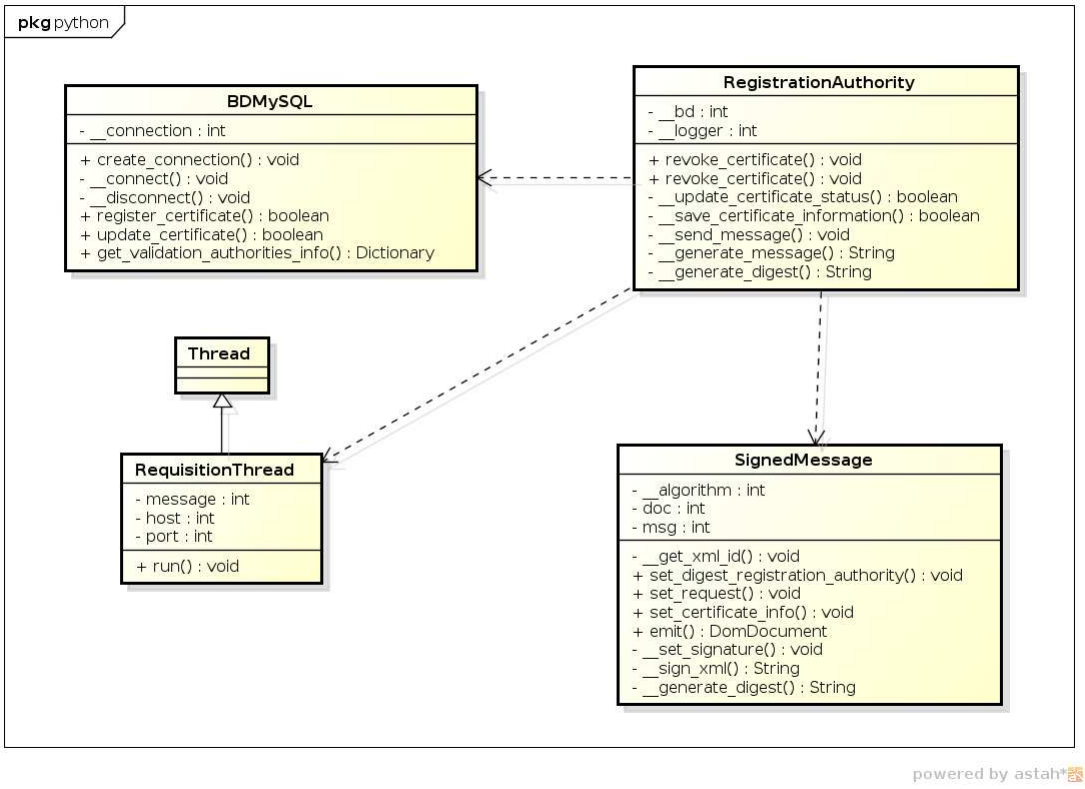

Figura 10: Diagrama de classes da Autoridade de Registro

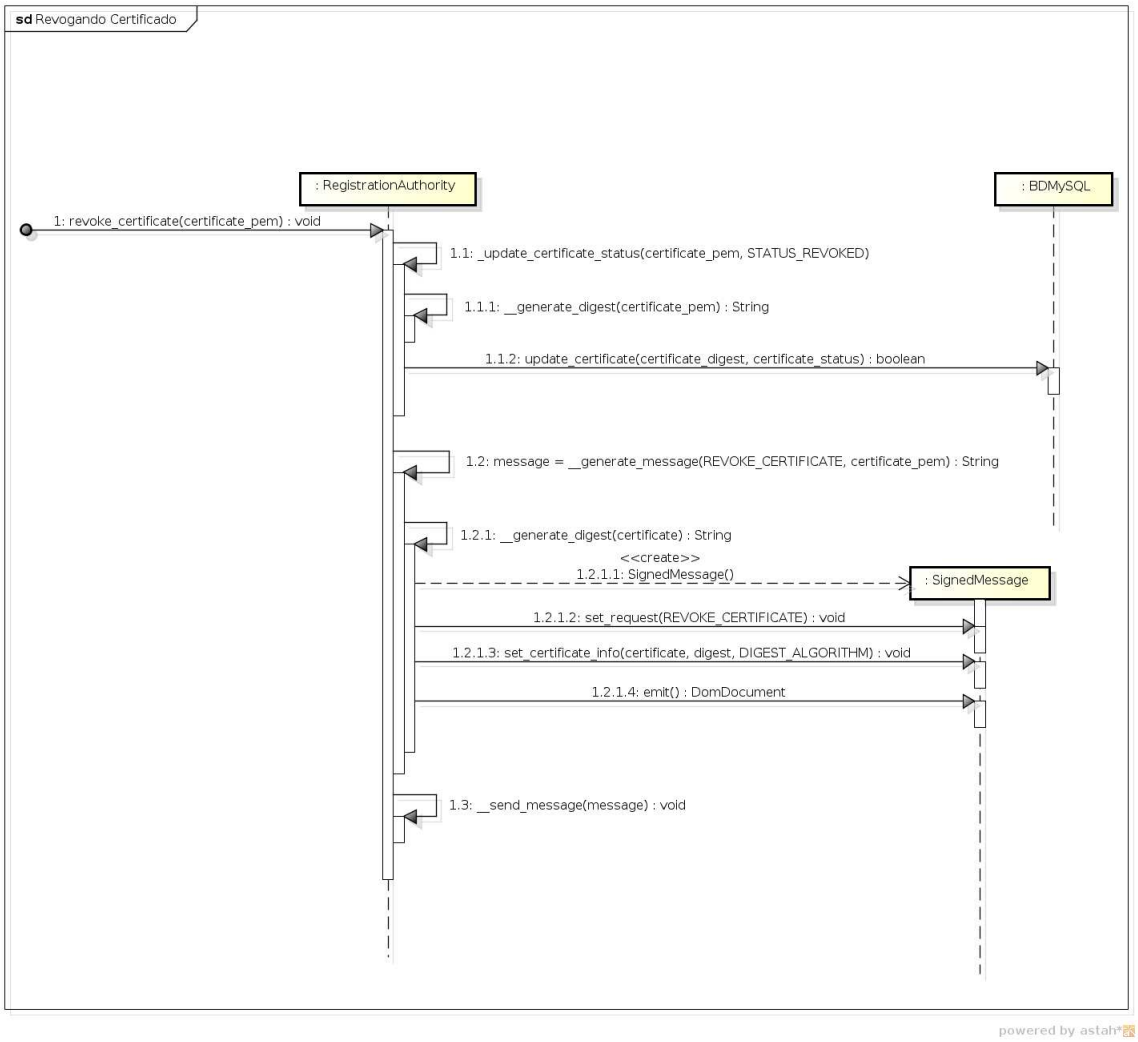

Figura 11: Diagrama de sequência de revogação

A Autoridade de Validação, após receber o xml assinado, verificará o tipo de pedido e atualizará sua base de dados. Nesse caso, o status do certificado em questão será alterado para revogado. É necessário assegurar que o banco de dados de todas as AVs seja consistente com a base de dados da AR, para isso foi criada uma thread que é lançada sempre que a tentativa de revogação ou cadastro de certificados não obtiver sucesso, para tentar em paralelo concluir a operação. Caso essa thread também não consiga, estas tentativas são anotadas em um log. Futuramente deve ser implementado um mecanismo que leia esses logs e tente refazer as operações que não obtiveram sucesso.

5.5 AUTORIDADE DE VALIDAÇÃO

A Autoridade de Validação é uma entidade que surge do modelo proposto por Moecke et al. [1]. Segundo o modelo, os certificados são autoassinados pelo usuário, não tendo mais a assinatura da Autoridade Certificadora. Para tornar um certificado confiável para terceiro, é necessário cadastrá-lo perante a Autoridade de Registro, que confirmará a identidade do seu detentor e cadastrará as informações do certificado na sua base de dados. A AR enviará os dados do certificado para a AV, para que esta também cadastre suas informações na sua base de dados,

tornando-a consistente com a base da AR. Toda vez que uma entidade deseja verificar a validade de um certificado, é necessário perguntar à Autoridade de Validação se ela reconhece aquele certificado como válido; como resposta recebe um token contendo o identificador do certificado, sua situação e a assinatura da AV. Além de validar certificados, a AV também é responsável por verificar a validade da assinatura de documentos. Para isto, é necessário enviar o identificador único do documento (hash), o identificador único do certificado (hash) e a assinatura. A AV então verifica se o certificado é válido e, caso seja, verifica se a assinatura foi gerada por aquele certificado e se bate com o hash do documento, provando que o documento não foi alterado.

Podemos então dizer que a Autoridade de Validação é responsável por emitir provas autoassinadas (tokens de resposta) que confirmam a validade de certificados e assinaturas.

5.5.1 *MODELAGEM PROPOSTA*

Para a implementação da Autoridade de Validação foram criadas as seguintes classes:

- ValidationAuthority
- BDMySQL
- ResponseToken
- RequisitionToken
- RequisitionThread
- Configurations

Conforme a Figura 12:

A classe *ValidationAuthority* equivale a Autoridade de Validação. Ela é responsável por emitir provas da validade de certificados e validar documentos assinados. Quando um sistema deseja saber se o certificado é válido, ele envia um token com um pedido de verificação de validade do certificado para a AV. A AV então verifica se o certificado está cadastrado na sua base de dados e qual a sua situação: se foi revogado, expirou ou se continua válido. Ela então envia um token de resposta (*ResponseToken*) codificado em ASN1 e assinado pela AV. O mesmo ocorre para verificar se uma assinatura é válida: o usuário envia o hash do documento, o hash do certificado que gerou a assinatura e a assinatura. A AV então verifica se o certificado está cadastrado na sua base de dados, se é válido, se a assinatura foi realmente gerada por aquele certificado e se o hash do documento bate com o hash assinado. Finalmente, gera um token de resposta (*ResponseToken*) também codificado em ASN1.

O token emitido pela Autoridade de Validação deve ser assinado pela AV, para ter credibilidade de que foi ela quem emitiu aquele token e que ele não foi alterado por terceiros. No modelo de token proposto em [1], a assinatura é um campo obrigatório. A biblioteca utilizada para gerar arquivos ASN.1 foi a pyASN1, que exigia que todos os campos estivessem corretamente valorados para emitir o token que seria posteriormente assinado. Mas sendo a assinatura um campo obrigatório, não era possível gerar o token para assiná-lo, pois este campo encontrava-se vazio. Foi necessário tornar este campo optativo e, só após gerar a assinatura de todos os outros campos, valorá-lo e assim poder montar o token.

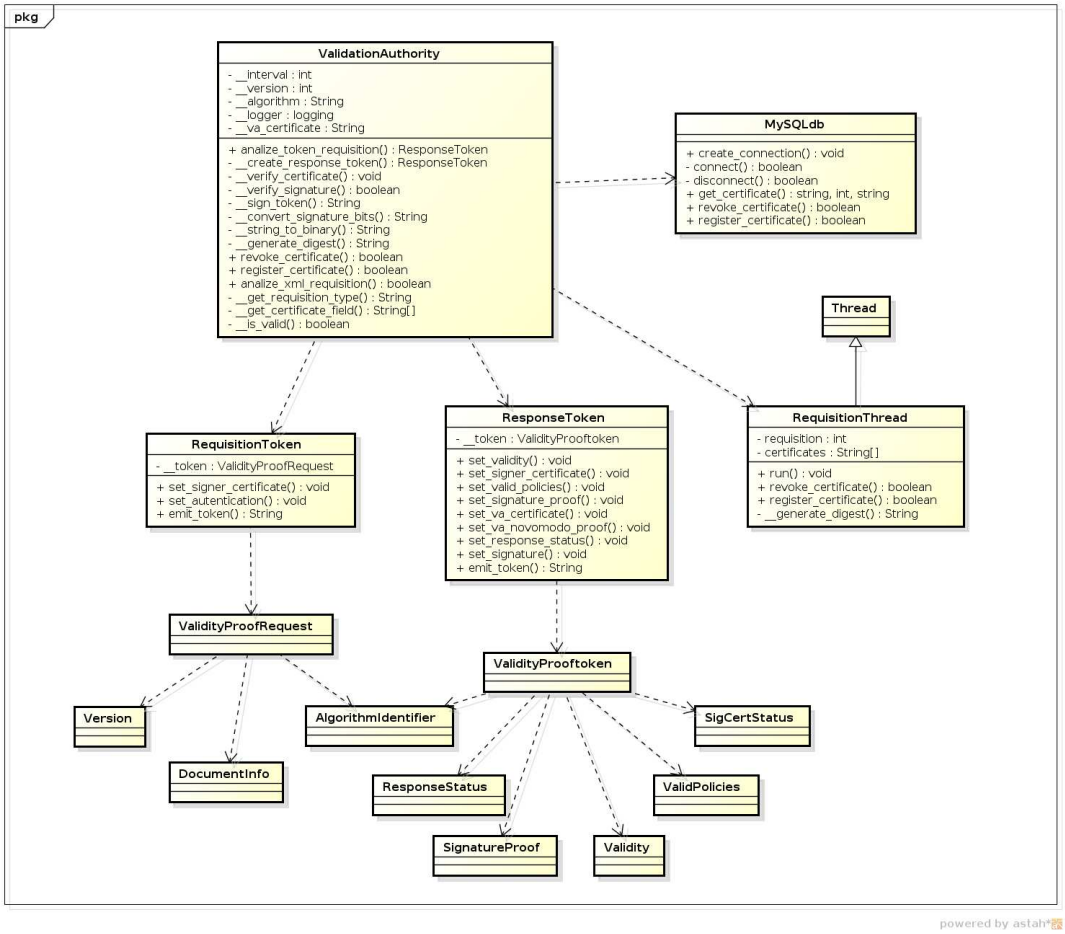

Figura 12: Diagrama de classes da Autoridade de Validação

Como algumas coisas não foram plenamente especificadas no artigo por serem questões ainda em discussão pelos autores, não foram completamente implementadas no projeto. Um exemplo disto, é o campo de políticas válidas (ValidPolicies) do token de resposta, sendo atribuído um valor fixo a ele devendo ser posteriormente alterado segundo o que for proposto por Cristian et al. O mesmo para o campo da prova de Novomodo, pois não foi utilizada uma Autoridade Certificadora, pré-requisito necessário para emitir a prova.

Outra característica da AV é que ela tem uma interface exclusiva com a Autoridade de Registro para receber pedidos de revogação e cadastro de certificados. Apenas a AR pode pedir para a AV executar essas operações, de forma que as informações sobre certificados sejam sempre consistentes entre as duas. Para isso, todo pedido feito pela AR é autoassinado e a AV verifica a assinatura para saber se aquele pedido partiu realmente da AR.

A classe *BDMySQL* faz o relacionamento com o banco de dados. Ele é responsável por permitir que a AV revoge e cadastre certificados, e recupere os dados relativos a determinado certificado para consulta posterior. Escolheu-se essa modelagem de forma que o Banco de Dados pudesse ser desacoplado da Autoridade de Validação e utilizado outro SGBD, como o Postgres, por exemplo, de forma que ficassem independentes.

As classes *ResponseToken* e *RequisitionToken* são responsáveis por gerar o token de resposta e de requisição, respectivamente. Ambas as classes geram tokens codificados em ASN.1 e tem uma estrutura bem definida, com todos os campos tendo seu formato e obrigatoriedade

especificados. Na Figura 13 podemos ver a estrutura do token de resposta e na Figura 14 a estrutura do token de requisição.

```
ValidityProoftoken ::= SEQUENCE {
    version              INTEGER { v1(1) },
    tokenValidity        Validity,
    userCertHashAlgorithm AlgorithmIdentifier
    userCertHash         OCTET STRING
    userSigCertStatus    SigCertStatus,
    validPolicies        ValidPolicies,
    signatureProof       SignatureProof OPTIONAL,
    vaCertificate        Certificate    OPTIONAL,
    vaNovomodoProof      BIT STRING     }
    responseStatus       ResponseStatus,
    signatureAlgorithm   AlgorithmIdentifier,
    signatureValue       BIT STRING     }

SigCertStatus ::= INTEGER {
    valid                (0),
    revoked              (1),
    expired              (2),
    error                (4)     }

ResponseStatus ::= INTEGER {
    okay                 (0),
    tooBusy              (1),
    invalidRequest       (2),
    internalError        (3),
    unsupportedVersion   (4),
    unknownCertificate   (5),
    badHashAlgorithm     (6)     }

ValidPolicies ::= SEQUENCE {
    policyIdentifier    OBJECT IDENTIFIER }

SignatureProof ::= SEQUENCE {
    hashAlgorithm   AlgorithmIdentifier
    signatureHash   BIT STRING   }

Validity ::= {
    notBefore   Time,
    notAfter    Time   }

AlgorithmIdentifier  ::=  SEQUENCE  {
     algorithm        OBJECT IDENTIFIER,
     parameters       ANY DEFINED BY algorithm
                                 OPTIONAL  }
```

Figura 13: Estrutura do Token de Resposta retirado de [1]

```
ValidityProofRequest ::= SEQUENCE {
    version           INTEGER { v1(1) },
    userCertHashAlgorithm AlgorithmIdentifier
    userCertHash       OCTET STRING
    document           DocumentInfo OPTIONAL }

DocumentInfo ::= SEQUENCE {
    hashAlgorithm      AlgorithmIdentifier,
    documentHash       OCTET STRING,
    documentSignature  BIT STRING         }
```

Figura 14: Estrutura do Token de Resquisição retirado de [1]

A classe *RequisitionThread* é uma thread lançada sempre que a operação de revogação ou registro de certificado não tiver sucesso, rodando em paralelo com o sistema. Essa thread será morta depois que exceder um determinado número de tentativas.

O arquivo *Configurations* contém todos os dados de configuração usuados no sistema como, por exemplo, o número máximo de tentativas para realizar determinada operação pela RequisitionThread. Esse arquivo foi criado para evitar que dados de configuração sejam inseridos diretamente no código, dificultando qualquer alteração no sistema e possibilitando que as configurações fossem mais dinâmicas.

A Figura 15 apresenta o diagrama de sequência mostrando o caminho principal de um pedido de validação e de autenticação da AV. Ela verifica o token recebido e faz as operações necessárias como verificar o certificado e a assinatura (caso seja um pedido de validação),

entre outras coisas. E como resposta monta um ResponseToken, com todas as informações necessárias para validar a assinatura ou autenticar o certificado.

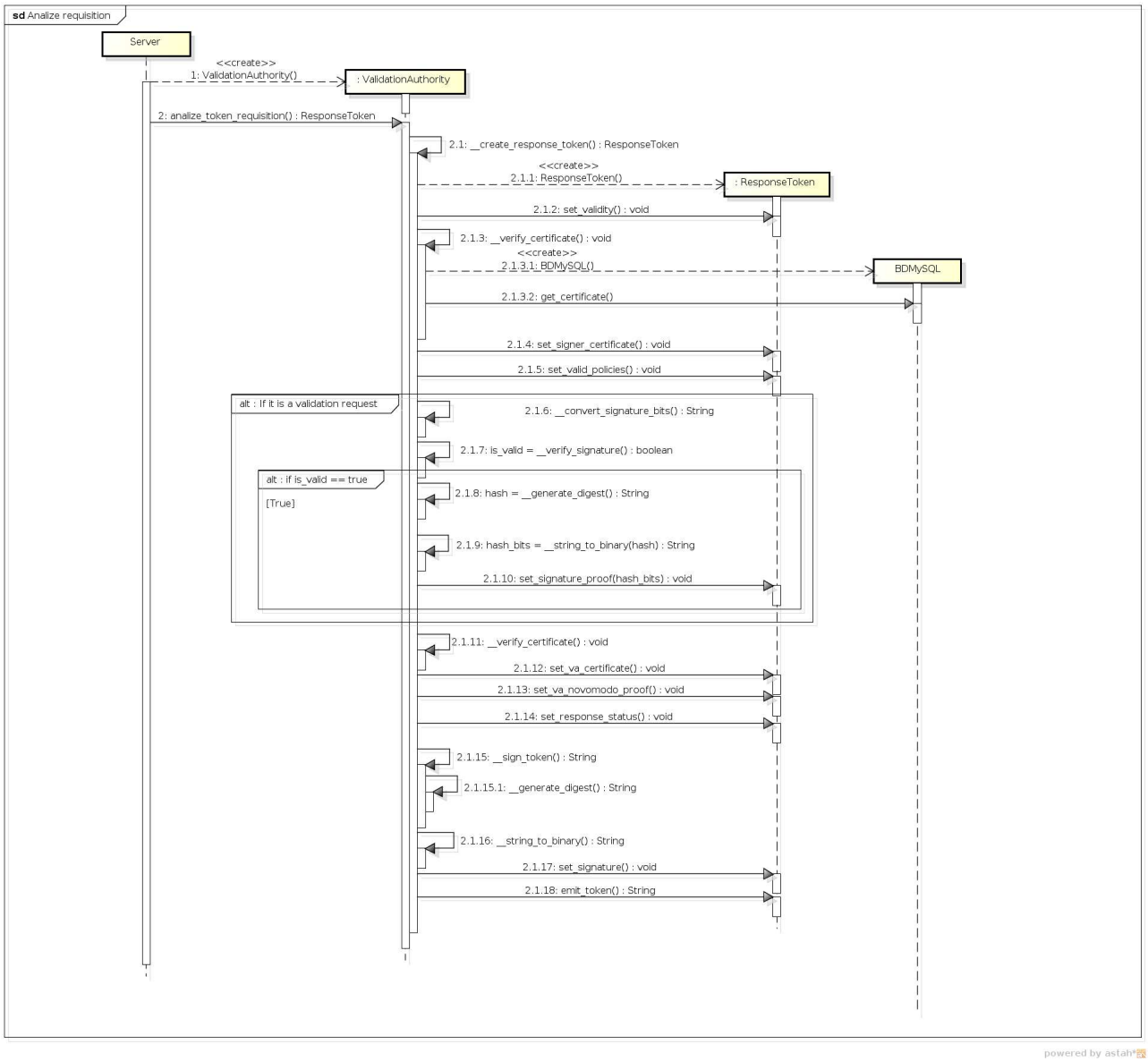

Figura 15: Diagrama de sequência mostrando o caminho principal da AV

5.6 VALIDADOR DE TOKEN

Tokens são emitidos pela Autoridade de Validação como resposta a uma requisição enviada. Requisição esta que pode ser um pedido de análise de validade de um certificado ou uma análise da validade de uma assinatura. Como os tokens são codificados em ASN.1, torna-se difícil para o usuário analisar o seu conteúdo de forma puramente visual. Para isto, foi criado um aplicativo que faz a interpretação do token, permitindo ao usuário verificar de forma simplificada qual foi a resposta final da AV sobre a sua requisição. Para isto, foi criada uma interface web onde é pedido que se entre com o Token e o certificado utilizado na operação com a AV. O Validador

analisa o Token, verificando se ele é válido e em seguida mostra o resultado na tela, conforme o esquema abaixo:

Status de resposta: a resposta da Autoridade de Validação referente ao pedido do usuário

- successful: operação de sucesso, tudo ocorreu corretamente
- invalidRequest: o pedido enviado foi inválido, no caso o token de requisição não foi corretamente especificado contendo campos inválidos
- internalError: algum erro interno ocorreu
- tryLater: seja por problemas no servidor, questões de manuntenção, pede-se que tente mais tarde
- unknownCertificate: o certificado enviado não foi encontrado na base de dados
- badDigestAlgorithm: o algoritmo de hash é inválido
- unsupportedVersion: a versão do token de requisição não é suportada
- invalidSignerSignature: a assinatura enviada é inválida, ou seja, não bate com o certificado enviado

Status do certificado do usuário: qual é a situação referente ao certificado enviado

- valid: o certificado é válido
- revoked: consta que o certificado foi revogado
- expired: consta que o certificado já expirou
- error: o certificado é inválido ou não foi encontrado na base de dados

A Figura 16 mostra a interface web do Analisador de Token:

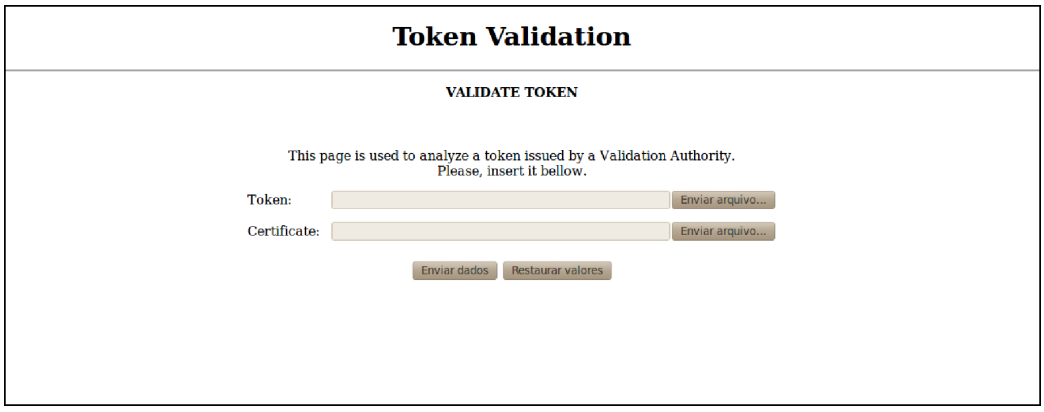

Figura 16: Página Web do Analisador de Token

5.7 BASE DE DADOS

Para a AV e a AR armazenarem os certificados validados e revogados por elas, foi criado um banco de dados que guarda toda informação relevante para as duas autoridades. Na Figura 17 temos a modelagem adotada:

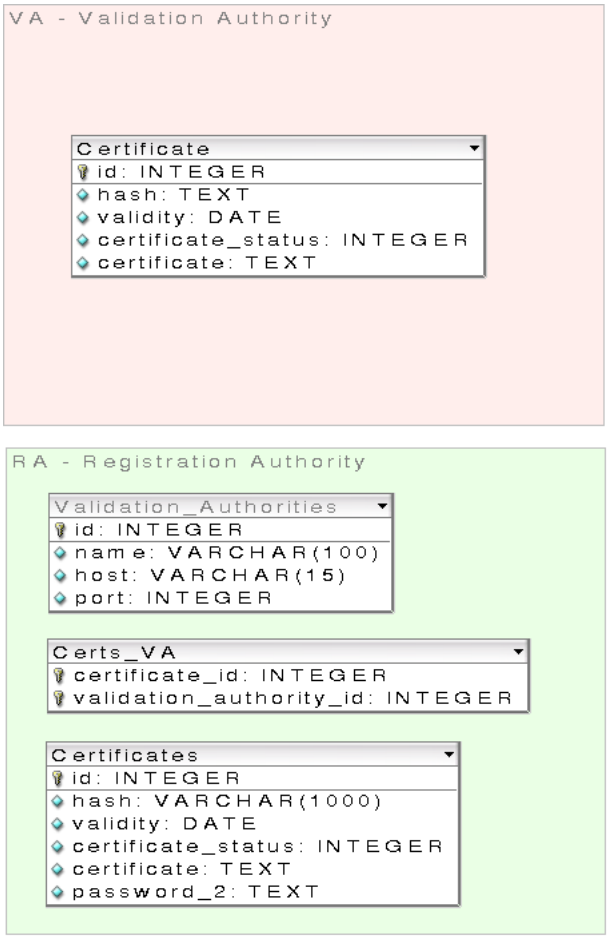

Figura 17: Modelagem dos bancos de dados utilizados

5.8 CONTRIBUIÇÕES AO MODELO PROPOSTO

Durante o desenvolvimento do protótipo, foram observadas algumas especificações ausentes no artigo que propõe o modelo, bem como pontos importantes que devem ser melhor analisados para a implementação prática do modelo.

Como resposta a um pedido de validação de assinatura enviado à AV, um *token* de resposta é emitido. O token contém o campo ResponseStatus que especifica qual a situação do pedido. De acordo com o foi proposto em [1], este campo pode assumir os seguintes valores: sucesso, servidor ocupado, requisição inválida, erro interno, versão não suportada, certificado desconhecido e algoritmo de digest inválido. Apenas com essas opções não era possível especificar quando a assinatura analisada era inválida. Para resolver esta questão, foi proposto que fosse aceito mais um valor: invalidSignerSignature, que foi implementado no presente trabalho.

Também percebeu-se que alguns aspectos não estavam plenamente especificadas no artigo, como por exemplo a comunicação entre a AR e a AV. Neste trabalho foi utilizado um xml assinado, como já descrito. A forma de comunicação e envio das mensagens ASN.1 entre o usuário e as Autoridades também está em aberto, e para o protótipo, optou-se por utilizar uma interface web. O artigo não apresenta, por exemplo, nenhuma forma de associar qual é a Autoridade de Validação responsável pela validação de um determinado certificado, em um contexto onde mais de uma Autoridade de Validação esteja operando.

5.9 COMO UTILIZAR O PROJETO

O projeto foi dividido em três partes: a Autoridade de Validação, a Autoridade de Registro e o Validador de Tokens.

Abaixo será explicado em detalhes como utilizar cada uma das partes do sistema.

5.9.1 *UTILIZANDO A AUTORIDADE DE VALIDAÇÃO*

Existem duas formas de se comunicar com a Autoridade de Validação: através do envio de um token de requisição e o recebimento de um token de resposta, ambos assinados, ou através do envio de um xml também assinado. O primeiro caso é utilizado para entidades que desejam interagir com a AV, já o segundo é utilizado exclusivamente pela Autoridade de Registro.

Foi desenvolvida uma página web que permite a troca de tokens com a AV. Para rodá-la é necessário ter o apache instalado e configurado para permitir scripts CGI (Common Gateway Interface). Também é necessário colocar os arquivos CGI e as páginas html no diretório configurado para o apache. O servidor da AV deve ser executado para poder atender as requisições enviadas através da página web. Feito isso, é só abrir a página no navegador.

A interface web chamará o script CGI que monta um token de requisição e o envia para o servidor da Autoridade de Validação. O servidor verifica o que foi recebido, se um token ou um xml, e o envia para a AV, que executa o que foi pedido e retorna um token de resposta. O servidor envia para o requisitante a resposta que é mostrada em uma página web com a opção de fazer o download do token. É aconselhável fazer seu download para poder analisá-lo através do Analisador de Token.

5.9.2 *UTILIZANDO A AUTORIDADE DE REGISTRO*

Para utilizar a Autoridade de Registro é necessário ter configurado o Apache da mesma maneira que a Autoridade de Validação. Diferentemente da AV não se utiliza xmls ou tokens assinados, mas sim de forma direta, apenas enviando uma requisição para o servidor da AR.

5.9.3 *UTILIZANDO O VALIDADOR DE TOKEN*

É necessário ter o Apache configurado da mesma maneira que a Autoridade de Validação e ter em mãos o token emitido pela AV. Feito isto, basta abrir a página do validador no navegador

e submeter o token, que será analisado por um script CGI. Os dados retirados da análise do token serão mostrados na tela do navegador.

6 CONSIDERAÇÕES FINAIS

O presente trabalho cumpriu com os seus objetivos, descrevendo o novo modelo de Infraestrutura de Chaves Públicas proposto em [1], e a sua implantação prática em protótipo. Essa nova abordagem de ICP foi implementada e com o uso e desenvolvimento do protótipo melhorias foram propostas à abordagem. A contribuição mais relevante do protótipo é possibilitar a verificação prática do funcionamento do modelo proposto bem como de aprimoramentos do mesmo.

A abordagem proposta em [1] apresenta-se como uma alternativa ao modelo mais utilizado de ICPs, o X.509. Através desta abordagem é possível simplificar bastante o processo de validação de uma assinatura digital.

Com a implementação de protótipos da Autoridade de Validação e Autoridade de Registro, pode-se afirmar que o modelo proposto por Moecke et al. [1] é viável e funcional. A implementação serve de base para a validação e experimentação do modelo, bem como permite avaliar aspectos importantes como o tamanho dos *tokens* e a quantidade de dados e processamento necessários para a operação da infraestrutura.

Foram ainda encontrados pontos da proposta que ainda precisam ser melhor especificados para viabilizar a implantação, onde correções foram sugeridas e implantadas.

6.1 TRABALHOS FUTUROS

O trabalho implementado é atualmente um protótipo podendo ser extendido a um trabalho mais formal. Como trabalho futuro, a gestão das provas de Novomodo pela Autoridade Certificadora também poderá ser implementada, com isto teríamos todas as Autoridades e o protótipo de uma ICP completa estaria desenvolvido.

Também será desejável usar um módulo para interface com HSM (Hardware Security Module) para aumentar a segurança do sistema, se possível através da integração com a biblioteca criptográfica desenvolvida pelo próprio LabSEC (LibCryptoSec), que já tem um módulo HSM implementado. Com isto, as operações como verificação de assinatura, assinatura de documentos e outras operações criptográficas seriam feitas dentro de um sistema fechado e seguro, aumentando a velocidade das operações e segurança do projeto como um todo.

Outro ponto importante é implementar um mecanismo que permita lançar threads para executar funções que não foram plenamente satisfeitas. Por exemplo, quando a Autoridade de Registro tenta enviar uma requisição para a Autoridade de Validação e não obtém resposta, ela lança uma thread rodando em paralelo que tenta executar essa operação um número determinado de vezes e, não obtendo sucesso, escreve os dados em um log. Atualmente não é feito nada

com esses dados enviados, é desejável criar uma thread que leia esses logs e refaça as operações em aberto.

Não foi possível implementar a parte de políticas de validade e a prova de Novomodo, pois a primeira não se encontra plenamente especificada no artigo tomado por base [1] e, para a segunda, é necessário uma comunicação com a Autoridade Certificadora, que não foi implementada. Posteriormente, será desejável implementar essas políticas assim que forem especificadas pelos autores do artigo e, como dito anteriormente, implementar o suporte em uma Autoridade Certificadora para emitir as provas de Novomodo.

REFERÊNCIAS

[1] MOECKE, C. T. et al. Uma ICP baseada em certificados digitais autoassinados. In: *Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*. Fortaleza: SBSEG, 2010. p. 91–104.

[2] WEISE, J. Public Key Infrastructure Overview. *Sun BluePrints Online*, ago. 2001.

[3] United States General Accounting Office. Advances and Remaining Challenges to Adoption of Public Key Infrastructure Technology. fev. 2001.

[4] 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). Disponível em: <http://www.ietf.org/rfc/rfc5280.txt>.

[5] GUTMANN, P. PKI Design for the Real World. *Department of Computer Science, University of Auckland*.

[6] DIFFIE, W.; HELLMAN, M. E. New directions in cryptography. *IEEE Transactions on Information Theory*, IT22, n. 6, 1976.

[7] FERGUSON, N.; SCHNEIER, B. *Practical Cryptography*. [S.l.]: Wiley Publishing, 2003.

[8] RSA Laboratories. *What is Diffie-Hellman?* Disponível em: <http://www.rsa.com/rsalabs/node.asp?id=2248>. Acesso em: Janeiro de 2011.

[9] Diffie-Hellman Overview. Disponível em: <http://cfsearching.blogspot.com/2010/09/my-first-look-at-diffie-hellman-merkle.html>. Acesso em: Janeiro de 2011.

[10] BURNETT, S.; PAINE, S. *Criptografia e Segurança*. [S.l.: s.n.], 2002.

[11] WANG, Y. *SPKI*. 7 out. 1998. Disponível em: <http://users.tkk.fi/yuwang/publications/SPKI/SPKI.html>.

[12] PLFLEEGER, C. P. *Security in Computing*. [S.l.: s.n.], 1996.

[13] NUNES, D. S. *Pki Public Key Infrastructure*. 4 nov. 2007. Disponível em: <http://www.gta.ufrj.br/grad/072/delio/index.html>.

[14] MOECKE, C. T. Assinatura digital de documentos eletrônicos na ICP-Brasil. *TCC, UFSC*, 2008.

[15] KLYNE, G.; NEWMAN, C. *Date and Time on the Internet: Timestamps*. IETF, jul. 2002. RFC 3339 (Proposed Standard). (Request for Comments, 3339). Disponível em: <http://www.ietf.org/rfc/rfc3339.txt>.

[16] GUTMANN, P. PKI: It's Not Dead, Just Resting. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 35, n. 8, p. 41–49, 2002. ISSN 0018-9162.

[17] RIVEST, R. L. Can We Eliminate Certificate Revocations Lists? In: *FC '98: Proceedings of the Second International Conference on Financial Cryptography*. London, UK: Springer-Verlag, 1998. p. 178–183. ISBN 3-540-64951-4.

[18] ADAMS, C. et al. *Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)*. IETF, ago. 2001. RFC 3161 (Proposed Standard). (Request for Comments, 3161). Disponível em: <http://www.ietf.org/rfc/rfc3161.txt>.

[19] ROSSUM, G. van. *Python*. Disponível em: <http://www.python.org/>. Acesso em: maio de 2011.

[20] TOIVONEN, H.; SIONG, N. P. *M2Crypto*. Disponível em: <http://chandlerproject.org/bin/view/Projects/MeTooCrypto>. Acesso em: Agosto de 2010.

[21] ActiveState. *Comparison of Python cryptography module*. 2010. Disponível em: <http://mikeivanov.com/pc/python-crypto.pdf>. Acesso em: Maio de 2011.

[22] MySQL. *MySQL*. Disponível em: <http://www.mysql.com/>. Acesso em: Agosto de 2010.

[23] DUSTMAN, A. *MySQL Python*. Disponível em: <http://mysql-python.sourceforge.net/>. Acesso em: Dezembro de 2010.

[24] Ilya Etingof. *PyASN1*. Disponível em: <http://pyasn1.sourceforge.net/>. Acesso em: Agosto de 2010.

[25] Apache Software Foundation. *Apache*. Disponível em: <http://www.apache.org/>. Acesso em: Agosto de 2010.

[26] Active State. *Komodo Edit*. Disponível em: <http://www.activestate.com/komodo-edit>. Acesso em: Agosto de 2010.

[27] FabForce. *DB Designer*. Disponível em: <http://fabforce.net/dbdesigner4/>. Acesso em: Maio de 2011.

[28] Astah. *Astah Community*. Disponível em: <http://astah.change-vision.com/en/product/astah-community.html>. Acesso em: Agosto de 2010.

A CÓDIGO FONTE DA AUTORIDADE DE VALIDAÇÃO

A.1 AUTORIDADE DE VALIDAÇÃO - VALIDATIONAUTHORITY.PY

```
1
2  #!/usr/bin/env python
3  from M2Crypto import X509, EVP, util
4  from M2Crypto.EVP import MessageDigest
5  from M2Crypto.X509 import X509Error #so se for pegar certificado invalido
6  from BDMySQL import BDMySQL, ExceptionCertificateNotRegistered
7  from Token.ResponseToken import ResponseToken
8  from RequisitionThread import RequisitionThread
9  from pyasn1.codec.der import decoder
10 from pyasn1.error import PyAsn1Error
11 from pyasn1.type import univ, useful
12 import os, datetime, struct, pickle, logging
13 from conf import *
14
15 class ValidationAuthority:
16
17     def __init__(self):
18         self.__va_certificate = open(os.getcwd()+"/Certificados/
               hash_certificado.sha512","rb").read()
19         self.__interval = 1 #in days
20         self.__version = 1
21         self.__algorithm = "sha512"
22
23         self.__logger = logging.getLogger('VA_LOG')
24         hdlr = logging.FileHandler(LOG_FILENAME)
25         formatter = logging.Formatter('%(asctime)s %(levelname)s %(message)
               s')
26         hdlr.setFormatter(formatter)
27         self.__logger.addHandler(hdlr)
28         self.__logger.setLevel(logging.DEBUG)
29
30         #sucessful(0), invalidRequest(1), internalError(2), tryLater(3),
31         #unknownCertificate(4), badDigestAlgorithm(5), unsupportedVersion
               (6)
32         self.__response_status = 0
33
34         #response status
35         self.__successful = 0
36         self.__invalidRequest = 1
37         self.__internalError = 2
38         self.__tryLater = 3
39         self.__unknownCertificate = 4
40         self.__badDigestAlgorithm = 5
```

```
41          self.__unsupportedVersion = 6
42          self.__invalidSignerSignature = 7
43
44          #digest algorithm OID
45          #0:rsa 1:DSA 2:md2 3:md5 4:sha1 5:sha224 6:sha256 7:sha384 8:sha512
46          self.__digest_algorithm = {"rsa":'1.2.840.113549.1.1.1',
47                                     "dsa":'1.2.840.10040.4.1',
48                                     "md2":'1.2.840.113549.2.2',
49                                     "md5":'1.2.840.113549.2.5',
50                                     "sha1":'1.3.14.3.2.26',
51                                     "sha224":'2.16.840.1.101.3.4.2.4',
52                                     "sha256":'2.16.840.1.101.3.4.2.1',
53                                     "sha384":'2.16.840.1.101.3.4.2.2',
54                                     "sha512":'2.16.840.1.101.3.4.2.3'}
55
56          self.__digest_name = {'1.2.840.113549.1.1.1':'rsa',
57                                '1.2.840.10040.4.1':'dsa',
58                                '1.2.840.113549.2.2':'md2',
59                                '1.2.840.113549.2.5':'md5',
60                                '1.3.14.3.2.26':'sha1',
61                                '2.16.840.1.101.3.4.2.4':'sha224',
62                                '2.16.840.1.101.3.4.2.1':'sha256',
63                                '2.16.840.1.101.3.4.2.2':'sha384',
64                                '2.16.840.1.101.3.4.2.3':'sha512'}
65
66      def analize_token_requisition(self, requisition_token):
67          '''
68          Emit a valid or an authentication token based on the requisition
               token
69          @param requisition_token: token sent by the user with all data
               needed, can
70          be a validation or an authentication request
71          @type requisition_token: RequisitionToken
72          @return: a response token
73          @rtype: ResponseToken
74          '''
75
76          #algorithm used by the user to generate the hash (digest) - convert
                tuple to string joined by '.'
77          signer_certificate_hash_algorithm = ''.join(str(i) + '.' for i in
               requisition_token[1][0])[:-1]
78
79          #hash of signer certificate
80          signer_certificate_hash = requisition_token[2]
81
82          #if it is a validation request
83          if len(requisition_token) == 4:
84
85              self.__logger.info("Receive a validation requisition from " +
                   str(signer_certificate_hash))
86
87              #algorithm used to generate hash - convert tuple to string
                   joined by '.'
88              document_hash_algorithm = ''.join(str(i) + '.' for i in
                   requisition_token[3][0][0])[:-1]
89
90              #hash of the document you want to verify the validity
91              document_hash = requisition_token[3][1]
```

```
92
93              #signature of the document in bits
94              document_signature_bits = requisition_token[3][2]
95
96              token = self.__create_response_token(
                    signer_certificate_hash_algorithm, signer_certificate_hash,
97                                          document_hash_algorithm,
                                                document_hash,
                                                document_signature_bits)
98
99          #if it is an authentication request
100         elif len(requisition_token) == 3:
101
102             self.__logger.info("Receive an authentication requisition from
                    " + str(signer_certificate_hash))
103
104             token = self.__create_response_token(
                    signer_certificate_hash_algorithm, signer_certificate_hash)
105
106         #invalid request
107         else:
108
109             self.__logger.info("Receive an invalid requisition from " + str
                    (signer_certificate_hash))
110
111             self.__response_status = self.__invalidRequest
112
113         return token
114
115
116     def __create_response_token(self, signer_certificate_digest_algorithm,
                signer_certificate_digest,
117                                 document_digest_algorithm = None, document_digest =
                                        None, document_signature_bits = None):
118         '''
119         Create a response token
120         @param signer_certificate_digest_algorithm: algorithm used to
                generate the digest of certificate
121         @param signer_certificate_digest: digest (hash) of certificate
122         @param document_digest_algorithm: algorithm used to generate the
                document digest
123         @param document_digest: digest (hash) of document
124         @param document_signature_bits: bits of signature of the document
125
126         @type signer_certificate_hash_algorithm: long, int, string
127         @type signer_certificate_hash: string
128         @type document_hash_algorithm: long, int, string (empty if it is a
                authentication request)
129         @type document_hash: string (empty if it is a authentication
                request)
130         @type document_signature_bits: string (empty if it is a
                authentication request)
131         @return: a response token
132         @rtype: ResponseToken
133         '''
134
135         #create an empty token containing just the version
136         token = ResponseToken(self.__version)
```

```
137
138         #calculate not_before and not_after (for how long this token will
                be valid)
139         if document_signature_bits is not None:
140             not_before = datetime.date.today()
141             interval = datetime.timedelta(days=self.__interval)
142             not_after = not_before + interval
143             token.set_validity(not_before, not_after)
144         else:
145             today = datetime.date.today()
146             token.set_validity(today, today)
147
148
149         #get all data about this certificate
150         cert, status, validity = self.__verify_certificate(
                signer_certificate_digest)
151
152         digest_name = self.__digest_name[
                signer_certificate_digest_algorithm]
153         #token.set_signer_certificate(self.__digest_algorithm[digest_name],
                 signer_certificate_digest, status )
154         scg = univ.OctetString(str(signer_certificate_digest).split("=")
                [1])
155         token.set_signer_certificate(self.__digest_algorithm[digest_name],
                scg, status )
156         token.set_valid_policies((1,1,1,1)) #TODO - definir politicas
157
158         #if its a validation request, verifiy if the signature is valid
159         if document_signature_bits is not None:
160             try:
161                 certificate = X509.load_cert_string(cert)
162                 document_signature = self.__convert_signature_bits(
                        document_signature_bits)
163
164                 is_valid = self.__verify_signature(certificate,
                        document_signature, document_digest, self.__digest_name[
                        document_digest_algorithm])
165                 if not is_valid:
166                     self.__response_status = self.__invalidSignerSignature
167
168                 hash = self.__generate_digest(document_signature, self.
                        __algorithm)
169                 hash_bits = self.__string_to_binary(hash)
170                 token.set_signature_proof(self.__digest_algorithm[self.
                        __algorithm], "'"+hash_bits+"'B")
171
172             except X509Error:
173                 self.__logger.error("X509Error: Invalid Certificate.")
174                 self.__response_status = self.__invalidRequest
175                 token.set_signature_proof(self.__digest_algorithm[self.
                        __algorithm], "'11111111'B") #o que eu ponho aqui se deu
                         erro? vazio mesmo?
176
177         #get validation authority certificate
178         va_certificate, va_status, va_validity = self.__verify_certificate(
                self.__va_certificate)
179         token.set_va_certificate(va_certificate)
```

```
180         token.set_va_novomodo_proof( "'11111111'B" ) #todo: tem que obter a
                prova da ac raiz
181         token.set_response_status(self.__response_status)
182 
183         #generate the hash of token, sign and set on token
184         proof = self.__sign_token(token, self.__algorithm)
185         proof_bits = self.__string_to_binary(proof)
186         oid_digest = self.__digest_algorithm[self.__algorithm]
187         token.set_signature(oid_digest,"'"+proof_bits+"'B")
188 
189         return token
190 
191     def __verify_certificate(self, certificate_hash):
192         '''
193         Return the status of this certificate, its validity and the pem
194         @param certificate_hash: hash of certificate you want to consult
195         @type certificate_hash: string
196         @return: certificate pem string, status and validity of certificate
197         @rtype: string, integer, string
198         '''
199         bd = BDMySQL()
200 
201         try:
202             #analize certificate throught certificate_hash and return all
                    cert related data
203             return bd.get_certificate(certificate_hash)
204 
205         except ExceptionCertificateNotRegistered:
206             self.__response_status = self.__unknownCertificate
207             self.__logger.error("ExceptionCertificateNotRegistered:
                    Certificate not registered in database.")
208             return "", 0, ""
209         except Exception as error:
210             self.__logger.error(error)
211             self.__response_status = self.__internalError
212             return "", 0, ""
213 
214     def __verify_signature(self, certificate, signature, digest,
            digest_algorith):
215         '''
216         Return if signature is valid or not
217         @param certificate: X509.X509
218         @param signature: string
219         @param hash: string
220         @param digest: string
221         @return: True or False
222         @rtype: boolean
223         '''
224         try:
225             #extract public key from certificate
226             pubkey = certificate.get_pubkey()
227 
228             #set the digest algorithm
229             pubkey.reset_context(md=digest_algorith)
230 
231             #initialize operation
232             pubkey.verify_init()
233 
```

```
234             #set hash
235             pubkey.verify_update(str(digest))
236 
237             #verify signature
238             result = pubkey.verify_final(signature)
239             return True if result == 1 else False
240 
241         except AttributeError as att_error:
242             self.__response_status = self.__unknownCertificate
243             self.__logger.error(att_error)
244             return False
245         except ValueError as value_error:
246             self.__logger.error(value_error)
247             if value_error == "unknown message digest":
248                 self.__response_status = self.__badDigestAlgorithm
249             return False
250         except Exception as error:
251             self.__logger.error(error)
252 
253     def __sign_token(self, token, digest):
254         '''
255         Sign a token with the private key of validation authority
256         @param token: token you want to sign
257         @type token: ResponseToken
258         @return signature of token
259         @rtype: string
260         '''
261         try:
262             privkey = EVP.load_key(os.getcwd()+"/Certificados/privkey.pem")
263             privkey.reset_context(md=digest)
264             privkey.sign_init()
265 
266             token_string = pickle.dumps(str(decoder.decode(token.emit_token
                    ())[0]))
267 
268             hash = self.__generate_digest(token_string, self.__algorithm)
269             privkey.sign_update(hash)
270             return privkey.sign_final()
271 
272         except AttributeError as att_error:
273             self.__response_status = self.__internalError
274             self.__logger.error(att_error)
275             return ""
276         except ValueError as value_error:
277             self.__response_status = self.__internalError
278             self.__logger.error(value_error)
279             return ""
280         except PyAsn1Error as pyans1_error:
281             self.__response_status = self.__internalError
282             self.__logger.error(pyans1_error)
283             return ""
284         except Exception as error:
285             self.__logger.error(error)
286 
287     def __convert_signature_bits(self, signature_bits):
288         '''
289         Convert the bits to ascii string
290         @param signature_bits: bits you want to convert to ascii
```

```
291         @type signature_bits: bits tuple
292         @return: signature converted
293         @rtype: string
294         '''
295         try:
296 
297             #how many 'bytes' will be necessary
298             size = len(signature_bits)/31
299 
300             #concatenate all bits in a string
301             bits = "".join([str(bit) for bit in signature_bits])
302 
303             v = ""
304             for i in range(size):
305                 #get a group of bits
306                 bit = bits[i*31:i*31+31]
307 
308                 #convert bit to 'byte' and concatenate
309                 v += struct.pack('I', int(bit, 2))
310 
311             #convert byte to ascii
312             return pickle.loads(v)
313 
314         except Exception as error:
315             self.__logger.error(error)
316             return ""
317 
318 
319 
320     def __string_to_binary(self, text):
321         '''
322         Convert ascii to binary string
323         @param text: string you want to convert
324         @type text: string
325         @return: string in binary
326         @rtype: string
327         '''
328         binary_text = ""
329         for char in text:
330             ascii = ord(char)
331             bin = []
332 
333             while (ascii > 0):
334                 if (ascii & 1) == 1:
335                     bin.append("1")
336                 else:
337                     bin.append("0")
338                 ascii = ascii >> 1
339 
340             bin.reverse()
341             binary = "".join(bin)
342             zerofix = (8 - len(binary)) * '0'
343             binary_text += zerofix + binary
344         return binary_text
345 
346     def __generate_digest(self, value, digest_alg):
347         '''
348         Generate in hex format the digest of a string
```

```
349         using the especif digest algorithm
350         @param value: string you want to generate the digest
351         @param digest_alg: digest algorithm you want to use
352         @type value: string
353         @type digest_alg: string
354         @return: hash of value in hex format
355         @rtype: string
356         '''
357 
358         md = EVP.MessageDigest(digest_alg)
359         md.update(value)
360         digest = md.final()
361         return hex(util.octx_to_num(digest))[2:-1].upper()
362 
363     #--------------------------------------------------------------------------
364 
365     def revoke_certificate(self, certificates_digest):
366         '''
367         Receive a list of certificates digest and revoke each one
368         @param certificates_digest: digest of each certificate you want to
                revoke
369         @type certificate_hash: list of string
370         @return: success or not
371         @rtype: bool
372         '''
373 
374         bd = BDMySQL()
375 
376         try:
377             #analize certificate throught certificate_hash and return all
                    cert related data
378             return bd.revoke_certificate(certificates_digest)
379 
380         except Exception as error:
381             self.__logger.error(error)
382 
383     def register_certificate(self, certificates_pem):
384         '''
385         Receive a list of certificates and register each one
386         @param certificates_pem: list of certificates in pem format
387         @type certificates_pem: list of string
388         @return: success or not
389         @type: bool
390         '''
391         try:
392 
393             certificate_digest = []
394             for cert in certificates_pem:
395                 certificate_digest.append(self.__generate_digest(cert, self
                        .__algorithm))
396 
397             bd = BDMySQL()
398 
399             #analize certificate throught certificate_hash and return all
                    cert related data
400             return bd.register_certificate(certificates_pem,
                    certificate_digest)
```

```
401
402             except Exception as error:
403                 self.__logger.error(error)
404                 return False
405
406
407         def analize_xml_requisition(self, xml_requisition):
408             '''
409             Analize a xml requisition and verify if
410             it is necessary to revoke or register these certificates and
411             do whatever is necessary
412             @param request: xml request with all data needed
413             @type request: dom document
414             @return: success or not
415             @rtype: bool
416             '''
417
418             #TODO: caso de alguma coisa errada, tem que refazer a operacao
419             #seria bom se desse pra agendar pra fazer de novo ou fazer outras
                    threads
420             #para nao ficar travado tentando refazer a operacao
421             try:
422                 valid = self.__is_valid(xml_requisition)
423                 if not valid: return False
424
425                 requisition = self.__get_requisition_type(xml_requisition)
426
427                 if requisition == REVOKE_REQUISITION:
428                     cert_digests = self.__get_certificate_field(xml_requisition
                            , "digest")
429                     resp = self.revoke_certificate(cert_digests)
430
431                     #if something went wrong, launch a thread to redo the work
432                     if not resp:
433                         thread = RequisitionThread(REVOKE_REQUISITION,
                                cert_digests)
434                         thread.start()
435                     return True
436
437                 elif requisition == REGISTER_REQUISITION:
438                     certs = self.__get_certificate_field(xml_requisition, "
                            certificate")
439                     resp = self.register_certificate(certs)
440
441                     #if something went wrong, launch a thread to redo the work
442                     if not resp:
443                         thread = RequisitionThread(REGISTER_REQUISITION, certs)
444                         thread.start()
445                     return True
446                 else:
447                     self.__logger.critical("Invalid Requisition Type.")
448                     return False
449
450             except Exception as error:
451                 self.__loger.error(error)
452                 return False
453
454         def __get_requisition_type(self, xml):
```

```
455             '''
456             Verify if the xml is a revoke or registration request
457             @param xml: xml you want to verify the requisition type
458             @type xml: dom document
459             @return: the field value
460             @rtype: string
461             '''
462             return xml.getElementsByTagName("request")[0].firstChild.data
463
464         def __get_certificate_field(self, xml, field):
465             '''
466             Get the value of a specific field from xml
467             @param xml: the specific xml
468             @type xml: dom document
469             @param field: the field you want to retrieve
470             @type field: string
471             @return: all values that match the respective field
472             @rtype: list
473             '''
474             certs = xml.getElementsByTagName("certificate_info")
475
476             digests = []
477             for cert in certs:
478                 element = cert.getElementsByTagName(field)
479                 for node in element:
480                     digests.append(node.firstChild.nodeValue)
481             return digests
482
483         def __is_valid(self, xml):
484             '''
485             Verify if this xml is properly formered and signed
486             @param xml: a request xml
487             @type xml: dom document
488             @return: if it is a valid xml
489             @rtype: bool
490             '''
491
492             #remove the <signature></signature> from xml
493             xml_split = xml.toprettyxml().split("<signature>")
494             xml_split1 = xml_split[-1].split("</signature>")[-1]
495             xml_string = xml_split[:-1][0][:-2] + xml_split1
496
497             #calculate the digest
498             digest = self.__generate_digest(xml_string, self.__algorithm)
499
500             #get the registration authority certificate
501             digest_ra = xml.getElementsByTagName("registration_authority")[0].
                    firstChild.data
502             certificate_string, status, validaty = self.__verify_certificate(
                    digest_ra)
503             certificate = X509.load_cert_string(certificate_string)
504
505             #get the signature
506             signature = xml.getElementsByTagName("signature")[0].firstChild.
                    data
507
508             return self.__verify_signature(certificate, signature, digest, self
                    .__algorithm)
```

A.2 CLASSE DO BANCO DE DADOS - BDMYSQL.PY

```
1
2  #!/usr/bin/env python
3
4  import MySQLdb
5  from conf import *
6  from MySQLdb import OperationalError
7
8  class BDMySQL:
9      '''
10     Connection with mysql db. Connect, disconnect and execute
11     database operations.
12     '''
13
14     def __init__(self):
15         self.__connection = None
16
17
18     def create_connection(f):
19         '''
20         DECORATOR: open an connection, call the especific function,
21         close the connection and return the value
22         '''
23         def new_f(self, *args):
24             self.__connect()
25             ret = f(self, *args)
26             self.__disconnect()
27             return ret
28         return new_f
29
30     def __connect(self):
31         '''
32         Try to connect in DB. If an error occur, an exception is raised.
33         '''
34         self.__connection = MySQLdb.connect(host = "localhost", user = "
               root", passwd = "toor", db = "VA")
35
36
37     def __disconnect(self):
38         '''
39         Try to disconnet from DB. If an error occur, an exception is raised
               .
40         If there is no connection opened, do nothing
41         '''
42         if self.__connection is not None:
43             self.__connection.close()
44
45
46     @create_connection
47     def get_certificate(self, digest):
48         '''
49         Get certificate pem string, certificate status and validity
```

```
50          of certificate you want to consult throught hash
51          @param digest: hash of certificate
52          @type digest: string
53          @return: certificate, status, validity
54          @rtype: string, int, string
55          @raise ExceptionCertificateNotRegistered: if this certificate isnt
                registered in db
56          '''
57
58          cursor = self.__connection.cursor()
59          query = "SELECT certificate_status, certificate, validity FROM
                Certificate WHERE hash = '" + str(digest) + "';"
60
61          cursor.execute(query)
62          result = cursor.fetchone()
63
64          #get certificate pem, status and validity of certificate
65          if result is not None:
66
67              status = int(result[0])
68              certificate = result[1]
69              validity = result[2]
70              return certificate, status, validity
71
72          #if this certificate is not registered in BD
73          else:
74              raise ExceptionCertificateNotRegistered
75
76      @create_connection
77      def revoke_certificate(self, digests):
78
79          '''
80          Revoke a list of certificates
81          @param digests: digests of certificates you want to revoke
82          @type digests: list of string
83          @return: success or not
84          @rtype: bool
85          '''
86
87          cursor = self.__connection.cursor()
88          query = "START TRANSACTION;"
89          for digest in digests:
90              query += "UPDATE Certificate SET certificate_status = " + str(
                    STATUS_REVOKED) + " WHERE hash = '" + str(digest) + "';"
91          query += "COMMIT;"
92          result = cursor.execute(query) #0 = true 1 = false
93
94          return True if result == 0 else False
95
96      @create_connection
97      def register_certificate(self, certificates_pem, certificates_digest):
98          '''
99          Register a list of certificates
100         @param certificates_pem: pem of certificates you want to register
101         @type certificates_pem: list of string
102         @param certificates_digest: digests of certificates you want to
                register
103         @type certificates_digest: list of string
```

```
104         @return: success or not
105         @rtype: bool
106         '''
107 
108         cursor = self.__connection.cursor()
109         query = "START TRANSACTION;"
110         i = 0
111         for certificate_pem in certificates_pem:
112             certificate_digest = certificates_digest[i]
113             i+=1
114             query = "INSERT INTO Certificate(hash, certificate_status,
                    certificate) VALUES('"+certificate_digest+"', "+str(
                    STATUS_VALID)+" ,'"+certificate_pem+"') ;"
115         query += "COMMIT;"
116         result = cursor.execute(query)
117 
118         return True if result == 0 else False
119 
120 
121 class ExceptionCertificateNotRegistered(Exception):
122     def __str__(self):
123         return repr("Certificate not registered in database.")
```

A.3 THREAD - REQUISITIONTHREAD.PY

```
1  from threading import Thread
2  from BDMySQL import BDMySQL
3  from M2Crypto import EVP, util
4  from conf import *
5  import sys
6 
7  class RequisitionThread(Thread):
8 
9      def __init__(self, requisition, certificates):
10         Thread.__init__(self)
11         self.requisition = requisition
12         self.certificates = certificates
13 
14 
15     def run ( self ):
16         resp = False
17         n = 0
18 
19         #keep trying till succeed or exceed the number of attempts
20         if self.requisition == "REVOKE_CERTIFICATE":
21             while not resp and n < ATTEMPTS:
22                 resp = self.revoke_certificate(self.certificates)
23                 n+=1
24         elif self.requisition == "REGISTER_CERTIFICATE":
25             while not resp and n < ATTEMPTS:
26                 resp = self.register_certificate(self.certificates)
27                 n+=1
28         sys.exit(0)
29 
30 
```

```
31      def revoke_certificate(self, certificates_digest):
32          '''
33          Receive a list of certificates digest and revoke each one
34          @param certificates_digest: digest of each certificate you want to
                revoke
35          @type certificate_hash: list of string
36          @return: success or not
37          @rtype: bool
38          '''
39
40          bd = BDMySQL()
41
42          return bd.revoke_certificate(certificates_digest)
43
44      def register_certificate(self, certificates_pem):
45          '''
46          Receive a list of certificates and register each one
47          @param certificates_pem: list of certificates in pem format
48          @type certificates_pem: list of string
49          @return: success or not
50          @type: bool
51          '''
52          print "registando certificado"
53          certificate_digest = []
54          for cert in certificates_pem:
55              certificate_digest.append(self.__generate_digest(cert,
                    DIGEST_ALGORITHM))
56
57          bd = BDMySQL()
58
59
60          return bd.register_certificate(certificates_pem, certificate_digest
                )
61
62
63      def __generate_digest(self, value, digest_alg):
64          '''
65          Generate in hex format the digest of a string
66          using the especif digest algorithm
67          @param value: string you want to generate the digest
68          @param digest_alg: digest algorithm you want to use
69          @type value: string
70          @type digest_alg: string
71          @return: hash of value in hex format
72          @rtype: string
73          '''
74
75          md = EVP.MessageDigest(digest_alg)
76          md.update(value)
77          digest = md.final()
78          return hex(util.octx_to_num(digest))[2:-1].upper()
```

B CÓDIGO FONTE DA AUTORIDADE DE REGISTRO

B.1 AUTORIDADE DE REGISTRO - REGISTRATIONAUTHORITY.PY

```
1
2  #!/usr/bin/env python
3
4  import time, socket, pickle, select
5  import logging
6  from M2Crypto import X509, EVP, util
7  from Configurations import *
8  from SignedMessage import SignedMessage
9  from RequisitionThread import RequisitionThread
10 from BDMySQL import BDMySQL
11 from MySQLdb import IntegrityError
12
13 # SE EU NAO CONSEGUIR ENVIAR PRA ALGUM HOST, MESMO ASSIM EU SALVO NO BANCO,
       CERTO?
14 # PARA PODER SIMPLESMENTE TENTAR MAIS TARDE
15 #
16 #
17 class RegistrationAuthority:
18
19     def __init__(self):
20         self.__bd = BDMySQL()
21
22         self.__logger = logging.getLogger('RA_LOG')
23         hdlr = logging.FileHandler(LOG_FILENAME)
24         formatter = logging.Formatter('%(asctime)s %(levelname)s %(message)
              s')
25         hdlr.setFormatter(formatter)
26         self.__logger.addHandler(hdlr)
27         self.__logger.setLevel(logging.DEBUG)
28
29     def revoke_certificate(self, certificate_pem):
30         '''
31         Notice all Validation Authorities that a specific
32         certificate was revoked
33         @param certificate_digest: unique certificate identifier (digest)
34         @type certificate_digest: string
35         '''
36         try:
37             self.__update_certificate_status(certificate_pem,
                  STATUS_REVOKED)
38             message = self.__generate_message(REVOKE_CERTIFICATE,
                  certificate_pem)
39             self.__send_message(certificate_pem, message)
```

```
40              return "Certified revoked successfully."
41
42          except Exception as msg:
43              self.__logger.error(msg)
44              return msg
45
46      def register_certificate(self, certificate_pem):
47          '''
48          Send a valid certificate to all Validation Authorities
49          to register the certificate in their DB
50          @param certificate_pem: pem of a valid certificate you want to
                register
51          @type certificate_pem: string
52          '''
53
54          try:
55              self.__save_certificate_information(certificate_pem)
56              message = self.__generate_message(REGISTER_CERTIFICATE,
                    certificate_pem)
57              self.__send_message(certificate_pem, message)
58              return "Certified registered successfully."
59
60          except Exception as msg:
61              self.__logger.error(msg)
62              return msg
63
64      def __update_certificate_status(self, certificate_pem,
            certificate_status):
65          '''
66          Update certificate status
67          @param certificate_pem: certificate you want to revoke
68          @type certificate_pem: string
69          @param certificate_status: status of certificate
70          @type certificate_status: integer
71          @return: success or not
72          @rtype: boolean
73          '''
74
75          certificate_digest = self.__generate_digest(certificate_pem)
76          return self.__bd.update_certificate(certificate_digest,
                certificate_status)
77
78
79      def __save_certificate_information(self, certificate_pem):
80          '''
81          Register in database a new certificate information
82          @param certificate_pem: certificate you want to register
83          @type certificate_pem: string
84          @return: success or not
85          @rtype: boolean
86          '''
87          certificate = X509.load_cert_string(certificate_pem)
88          validity = certificate.get_not_after()
89          certificate_digest = self.__generate_digest(certificate_pem)
90          certificate_status = STATUS_VALID
91
92          return self.__bd.register_certificate(certificate_pem,
                certificate_digest, validity, certificate_status)
```

```
93
94
95
96      def __send_message(self, certificate_pem, message):
97          '''
98          Send a signed xml request to all AC registered
99          @param message: xml with all needed data
100         @type message: string
101         '''
102
103         #establish a connect with each server and send the message
104         self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
105
106         #get all va's related to this certificate
107         validation_authorities_ip = self.__bd.
                get_validation_authorities_info(certificate_pem)
108
109         for host in validation_authorities_ip.keys():
110             port = int(validation_authorities_ip[host])
111             try:
112
113                 self.server.connect((host,port))
114                 self.server.send(pickle.dumps(message))
115                 self.server.setblocking(0)
116                 ready = select.select([self.server], [], [],
                        TIMEOUT_IN_SECONDS)
117
118                 if ready[0]:
119                     resp = self.server.recv(4096)
120                     if not resp:
121                         print("Not possible to send to host ", host, " in
                                port ", port)
122                     else:
123                         self.__logger.info("Success in host ", host, " in
                                port ", port)
124
125                 #didnt receive response from server
126                 else:
127                     self.__logger.warning("Launch requisition thread to
                            host ", host, " in port ", port)
128                     thread = RequisitionThread(message, host, port)
129                     thread.start()
130
131             #if it was not possible to send to this host...
132             except Exception as e:
133                 self.__logger.critical("Not possible to reach host " + str(
                        host) + " in port " + str(port))
134
135         #close connection
136         self.server.close()
137
138     def __generate_message(self, request, certificate):
139         '''
140         Generate a signed xml with a specific request
141         @param request: type of requisition (revoke, register, etc)
142         @type request: string
143         @param certificate: specific certificate (the one you want to
                revoke, register, etc)
```

```
144         @type certificate: string
145         '''
146
147         digest = self.__generate_digest(certificate)
148         message = SignedMessage()
149         message.set_request(request)
150         message.set_certificate_info(certificate, digest, DIGEST_ALGORITHM)
151         return message.emit()
152
153     def __generate_digest(self, value):
154         '''
155         Generate in hex format the digest of a string
156         using the especif digest algorithm
157         @param value: string you want to generate the digest
158         @type value: string
159         @return: hash of value in hex format
160         @rtype: string
161         '''
162         md = EVP.MessageDigest(DIGEST_ALGORITHM)
163         md.update(value)
164         digest = md.final()
165         return hex(util.octx_to_num(digest))[2:-1].upper()
```

B.2 CLASSE DO BANCO DE DADOS - BDMYSQL.PY

```
1  #!/usr/bin/env python
2
3  import MySQLdb
4  from MySQLdb import OperationalError, IntegrityError
5  from Configurations import *
6
7  class BDMySQL:
8
9      def __init__(self):
10         self.__connection = None
11
12     def create_connection(f):
13         '''
14         DECORATOR: open an connection, call the especific function,
15         close the connection and return the value
16         '''
17         def new_f(self, *args):
18             self.__connect()
19             ret = f(self, *args)
20             self.__disconnect()
21             return ret
22         return new_f
23
24     def __connect(self):
25         '''
26         Try to connect in DB. If it is not possible, raise an exception.
27         '''
28         self.__connection = MySQLdb.connect(host = "localhost", user = "
               root", passwd = "toor", db = "RA")
29
```

```
30      def __disconnect(self):
31          '''
32          Try to disconnet from DB. If there is no connection opened, do
                nothing.
33          If it was not possible to close an opened connection, raise an
                exception.
34          '''
35          if self.__connection is not None:
36              self.__connection.close()
37
38      @create_connection
39      def register_certificate(self, certificate_pem, certificate_digest,
            validity, certificate_status):
40          '''
41          Register a certificate in database.
42          If an error occur, an exception is raised.
43          @param certificate_pem: certificate string
44          @type certificate_pem: string
45          @param certificate_digest: certificate hash
46          @type certificate_digest: string
47          @param validity: certificate validity
48          @type validity: string
49          @param certificate_status: status certificate
50          @type certificate_status: integer
51          @return: success or not
52          @rtype: boolean
53          '''
54
55          #insert new certificate in database
56          cursor = self.__connection.cursor()
57          query = "INSERT INTO Certificates(hash, certificate_status,
                certificate) VALUES('"+certificate_digest+"', "+str(STATUS_VALID
                )+" ,'"+certificate_pem+"') ;"
58          result = cursor.execute(query)
59
60          #select id certificate that has just been inserted
61          query = "SELECT C.id FROM Certificates C WHERE hash = '" + str(
                certificate_digest) + "' ;"
62          cursor.execute(query)
63          ret = cursor.fetchone()
64
65          #if there is an id
66          if ret is not None:
67              cert_id = int(ret[0])
68
69              #retrieve all VAs ids
70              query = "SELECT VA.id FROM Validation_Authorities VA;"
71              cursor.execute(query)
72              va_ids = cursor.fetchall()
73
74              #foreach VA, insert in table the relation VA-Cert
75              query = "START TRANSACTION;"
76              for va_id in va_ids:
77                  query += "INSERT INTO Certs_VA(certificate_id,
                        validation_authority_id) VALUES("+str(cert_id)+" , "+str
                        (int(va_id[0]))+"); "
78              query += "COMMIT;"
79              result = cursor.execute(query) #0 = true 1 = false
```

```
80
81              #if all run well, return true
82              return True if result == 0 else False
83
84          else:
85              return False
86
87      @create_connection
88      def update_certificate(self, certificate_digest, certificate_status):
            '''
89          Update certificate status. If an error occur, an exception is
               raised.
90          @param certificate_digest: hash of the certificate you want to
               update
91          @type certificate_digest: string
92          @param certificate_status: new status certificate you want to
               update
93          @type certificate_status: integer
94          @return sucess or not
95          @rtype: boolean
96          '''
97
98          cursor = self.__connection.cursor()
99
100         query = "UPDATE Certificates SET certificate_status = " + str(
               STATUS_REVOKED) + " WHERE hash = '" + str(certificate_digest) +
               "';"
101
102         result = cursor.execute(query)
103         return True if result == 1 else False
104
105
106     @create_connection
107     def get_validation_authorities_info(self, certificate_pem):
            '''
108         Retrieve from db all validation authorities relate to this
               certificate.
109         f an error occur, an exception is raise.
110         @return: a dictionary with all validation authorities host and port
111         @rtype: dictionary
112         '''
113
114         cursor = self.__connection.cursor()
115         query = "SELECT host, port FROM Validation_Authorities WHERE id IN
               " \
116         "(SELECT validation_authority_id FROM Certs_VA WHERE certificate_id
                = " \
117         "( SELECT id FROM Certificates WHERE certificate = '" +
               certificate_pem + "'));"
118
119         cursor.execute(query)
120         result = cursor.fetchall()
121         validation_authorities_ip = {}
122
123         if len(result) == 0:
124             raise ExceptionCertificateNotRegistered
125
126         for ip in result:
127
128
```

```
129             validation_authorities_ip[ ip[0] ] = int(ip[1])
130 
131         return validation_authorities_ip
132 
133 class ExceptionConnectionNotEstablished(Exception):
134     def __str__(self):
135         return repr("Connection not estabilish.")
136 
137 class ExceptionCertificateNotRegistered(Exception):
138     def __str__(self):
139         return repr("EXCEPTION: Certificate not registered.")
```

B.3 THREAD - REQUISITIONTHREAD.PY

```
1  #!/usr/bin/env python
2  from threading import Thread
3  from Configurations import *
4  import socket, pickle, sys, time, select
5  
6  
7  class RequisitionThread(Thread):
8  
9      def __init__(self, message, host, port):
10         Thread.__init__(self)
11         self.message = message
12         self.host = host
13         self.port = port
14 
15 
16     def run(self):
17         resp = False
18         attempts = 0
19 
20         #while there is no answer from Server
21         try:
22             while (not resp or resp == None) and attempts < ATTEMPTS_NUMBER
                   :
23                 server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
24                 server.connect((self.host,self.port))
25                 server.send(pickle.dumps(self.message))
26 
27                 ready = select.select([server], [], [], TIMEOUT_IN_SECONDS)
28                 if ready[0]:
29                     resp = server.recv(10000)
30                 else:
31                     attempts += 1
32         except socket.error: #host not reached - connection refused
33             sys.exit(0)
34 
35         except Exception as e:
36             print(type(e))
37 
38         sys.exit(0)
```

C TOKENS

C.1 TOKEN DE REQUISIÇÃO - REQUISITIONTOKEN.PY

```
#!/usr/bin/env python

from pyasn1.type import univ, namedtype, namedval, tag, constraint
from pyasn1.codec.ber import encoder, decoder

"""

ValidityProofRequest ::= SEQUENCE {
        version                                 INTEGER { v1(1) },
        signerCertificateDigestAlgorithm  AlgorithmIdentifier
        signerCertificateDigest             OCTET STRING
        document DocumentInfo                 OPTIONAL
}

DocumentInfo ::= SEQUENCE {
        digestAlgorithm          AlgorithmIdentifier,
        documentDigest           OCTET STRING,
        documentSignature        BIT STRING
}


"""

class Version(univ.Integer, object): pass

class AlgorithmIdentifier(univ.Sequence):
    componentType = namedtype.NamedTypes(
        namedtype.NamedType('algorithm',univ.ObjectIdentifier()),
        namedtype.OptionalNamedType('parameters', univ.Null())
       # namedtype.OptionalNamedType('parameters',)
    )

class DocumentInfo(univ.Sequence):
    componentType = namedtype.NamedTypes(
    namedtype.NamedType('digestAlgorithm', AlgorithmIdentifier()),
    namedtype.NamedType('documentDigest', univ.OctetString()),
    namedtype.NamedType('documentSignature', univ.BitString())
    )

class ValidityProofRequest(univ.Sequence, object):
    componentType = namedtype.NamedTypes(
    namedtype.NamedType('version', Version()),
```

```
43      namedtype.NamedType('signerCertificateDigestAlgorithm',
            AlgorithmIdentifier()),
44      namedtype.NamedType('signerCertificateDigest', univ.OctetString()),
45      namedtype.OptionalNamedType('document', DocumentInfo())
46      )
47
48  #end of asn1
49
50  class RequisitionToken():
51
52      def __init__(self, version):
53          self.__token = ValidityProofRequest()
54          self.__token.setComponentByPosition(0, version)
55
56      def set_signer_certificate(self, signer_certificate_digest_algorithm,
            signer_certificate_digest):
57          alg_ident = AlgorithmIdentifier()
58          alg_ident.setComponentByPosition(0,
                signer_certificate_digest_algorithm)
59          self.__token.setComponentByPosition(1,alg_ident)
60          self.__token.setComponentByPosition(2, signer_certificate_digest)
61
62      def set_autentication(self, digest_algorithm, document_digest,
            document_signature):
63          docInfo = DocumentInfo()
64          alg_ident = AlgorithmIdentifier()
65          alg_ident.setComponentByPosition(0,digest_algorithm)
66          docInfo.setComponentByPosition(0,alg_ident)
67          docInfo.setComponentByPosition(1,document_digest)
68          docInfo.setComponentByPosition(2,document_signature)
69          self.__token.setComponentByPosition(3,docInfo)
70
71      def emit_token(self):
72          return encoder.encode(self.__token)
```

C.2 TOKEN DE RESPOSTA - RESPONSETOKEN.PY

```
1   #!/usr/bin/env python
2   from pyasn1.type import univ, namedtype, namedval, tag, constraint, useful
3   from pyasn1.codec.der import decoder, encoder
4
5
6
7
8   class SigCertStatus(univ.Integer, object):
9       namedValues = namedval.NamedValues(
10          ('valid', 0),
11          ('revoked', 1),
12          ('expired', 2),
13          ('error', 3),
14          ('invalidSignature',4)
15      )
16
17  class ResponseStatus(univ.Integer):
18      namedValues = namedval.NamedValues(
```

```
19          ('successful', 0),
20          ('invalidRequest', 1),
21          ('internalError', 2),
22          ('tryLater', 3),
23          ('unknownCertificate', 4),
24          ('badDigestAlgorithm', 5),
25          ('unsupportedVersion', 6)
26      )
27
28  class ValidPolicies(univ.Sequence):
29      componentType = namedtype.NamedTypes(
30          namedtype.NamedType('policyIdentifier', univ.ObjectIdentifier())
31      )
32
33  class AlgorithmIdentifier(univ.Sequence):
34      componentType = namedtype.NamedTypes(
35          namedtype.NamedType('algorithm',univ.ObjectIdentifier()),
36          namedtype.OptionalNamedType('parameters', univ.Null())
37      )
38
39  class SignatureProof(univ.Sequence):
40      componentType = namedtype.NamedTypes(
41          namedtype.NamedType('digestAlgorithm', AlgorithmIdentifier()),
42          namedtype.NamedType('signatureHash', univ.BitString())
43      )
44
45  class Validity(univ.Sequence):
46      componentType = namedtype.NamedTypes(
47          namedtype.NamedType('notBefore', useful.UTCTime()),
48          namedtype.NamedType('notAfter', useful.UTCTime())
49      )
50
51
52
53  class ValidityProoftoken(univ.Sequence, object):
54      componentType = namedtype.NamedTypes(
55          namedtype.NamedType('version', univ.Integer()),
56          namedtype.NamedType('tokenValidity', Validity()),
57          namedtype.NamedType('signerCertificateDigestAlgorithm',
                AlgorithmIdentifier()),
58          namedtype.NamedType('signerCertificateDigest', univ.OctetString()),
59          namedtype.NamedType('signerCertificateStatus', SigCertStatus()),
60          namedtype.NamedType('validPolicies', ValidPolicies()),
61          namedtype.OptionalNamedType('signatureProof', SignatureProof()),
62          namedtype.OptionalNamedType('vaCertificate', univ.OctetString()),
63          namedtype.NamedType('vaNovomodoProof', univ.BitString()),
64          namedtype.NamedType('responseStatus', ResponseStatus()),
65          namedtype.OptionalNamedType('signatureAlgorithm',
                AlgorithmIdentifier()),
66          namedtype.OptionalNamedType('signatureValue', univ.BitString())
67      )
68
69  class ResponseToken():
70
71      def __init__(self, version):
72          self.__token = ValidityProoftoken()
73          self.__token.setComponentByPosition(0, version)
74
```

```
75      def set_validity(self, not_before, not_after):
76          validity = Validity()
77          validity.setComponentByPosition(0, not_before)
78          validity.setComponentByPosition(1, not_after)
79          self.__token.setComponentByPosition(1,validity)
80
81      def set_signer_certificate(self, signer_certificate_digest_algorithm,
            signer_certificate_digest, signer_certificate_status):
82          alg_ident = AlgorithmIdentifier()
83          alg_ident.setComponentByPosition(0,
               signer_certificate_digest_algorithm)
84          self.__token.setComponentByPosition(2, alg_ident)
85          self.__token.setComponentByPosition(3, signer_certificate_digest)
86          self.__token.setComponentByPosition(4, signer_certificate_status)
87
88      def set_valid_policies(self,valid_policies):
89          policies = ValidPolicies()
90          policies.setComponentByPosition(0, valid_policies)
91          self.__token.setComponentByPosition(5, policies)
92
93      def set_signature_proof(self,digest_algorithm, signature_hash):
94          signature_proof = SignatureProof()
95          alg_ident = AlgorithmIdentifier()
96          alg_ident.setComponentByPosition(0,digest_algorithm)
97          signature_proof.setComponentByPosition(0, alg_ident)
98          signature_proof.setComponentByPosition(1, signature_hash)
99          self.__token.setComponentByPosition(6, signature_proof)
100
101     def set_va_certificate(self, va_certificate):
102         self.__token.setComponentByPosition(7, va_certificate)
103
104     def set_va_novomodo_proof(self, va_novomodo_proof):
105         self.__token.setComponentByPosition(8, va_novomodo_proof)
106
107     def set_response_status(self, response_status):
108         self.__token.setComponentByPosition(9, response_status)
109
110     def set_signature(self, signature_algorithm, signature_value):
111         alg_ident = AlgorithmIdentifier()
112         alg_ident.setComponentByPosition(0,signature_algorithm)
113         self.__token.setComponentByPosition(10, alg_ident)
114         self.__token.setComponentByPosition(11, signature_value)
115
116     def emit_token(self):
117         return encoder.encode(self.__token,defMode=True, maxChunkSize=8)
```

D VALIDADOR DE TOKEN

D.1 VALIDADOR DE TOKENS - TOKENVALIDATION.PY

```
#!/usr/bin/env python
import pickle
from pyasn1.codec.ber import decoder
from M2Crypto import X509, EVP, util


def convert_signature_bits(signature_bits):
        '''
        Convert the bits to ascii string
        @param signature_bits: bits you want to convert to ascii
        @type signature_bits: bits tuple
        @return: signature converted
        @rtype: string
        '''

        size = len(signature_bits)/8

        bits = "".join([str(bit) for bit in signature_bits if bit!=","])
        v = ""
        for i in range(size):
            bit = bits[i*8:i*8+8]
            v += chr(int(bit,2))

        return v

def validate_token(token, certificate_string):
    if (token.find("setComponentByPosition(11") > 0):
        index = 10
    else:
        index = 9

    splits = str(token).split(".setComponentByPosition(" + str(index) +
       ",")[0]
    start = token.rfind("(") + 1
    end = token.rfind(")")
    signature_bits = token[start:end]
    lista = [ int(i) for i in signature_bits if (i!=")" and i != ',' and i
       != '\\' and i != " " and i!="\"" and i!="\'" and i!="B")]
    signature = convert_signature_bits(lista)
    md = EVP.MessageDigest("sha512")
    token_string = pickle.dumps(str(splits))
    md.update(token_string)
    digest = md.final()
```

```
42      hash = hex(util.octx_to_num(digest))[2:-1].upper()
43      certificate = X509.load_cert_string(certificate_string)
44      pubkey = certificate.get_pubkey()
45      pubkey.reset_context(md="sha512")
46      pubkey.verify_init()
47      pubkey.verify_update(str(hash))
48      result = pubkey.verify_final(signature)
49
50      return True if result == 1 else False
51
52  def get_response_status(token):
53      if (token.find("setComponentByPosition(11") > 0):
54          index = 8
55      else:
56          index = 7
57
58      splits = str(token).split(".setComponentByPosition(" + str(index) +
            ",")[1]
59      if splits.find("Integer('0')") > 0:
60          return "Successful"
61      elif splits.find("Integer('1')") > 0:
62          return "Invalid Request"
63      elif splits.find("Integer('2')") > 0:
64          return "Internal Error"
65      elif splits.find("Integer('3')") > 0:
66          return "Try Later"
67      elif splits.find("Integer('4')") > 0:
68          return "Unkown Certificate"
69      elif splits.find("Integer('5')") > 0:
70          return "Bad Digest Algorithm"
71      elif splits.find("Integer('6')") > 0:
72          return "Unsupported Version"
73
74  def get_sign_cert_status(token):
75      if (token.find("setComponentByPosition(11") > 0):
76          index = 3
77      else:
78          index = 2
79
80      splits = str(token).split(".setComponentByPosition(" + str(index) +
            ",")[1]
81      if splits.find("Integer('0')") > 0:
82          return "Valid"
83      elif splits.find("Integer('1')") > 0:
84          return "Revoked"
85      elif splits.find("Integer('2')") > 0:
86          return "Expired"
87      elif splits.find("Integer('3')") > 0:
88          return "Error"
89      elif splits.find("Integer('4')") > 0:
90          return "Invalid Signature"
```