UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Gustavo Souza Banegas, Vicente Silveira Inácio

FRAMEWORK LIBPKI: UM ARCABOUÇO PARA DESENVOLVIMENTO DE APLICAÇÕES UTILIZADAS NO ÂMBITO DE INFRAESTRUTURAS DE CHAVES PÚBLICAS.

Florianópolis

2012

Gustavo Souza Banegas, Vicente Silveira Inácio

FRAMEWORK LIBPKI: UM ARCABOUÇO PARA DESENVOLVIMENTO DE APLICAÇÕES UTILIZADAS NO ÂMBITO DE INFRAESTRUTURAS DE CHAVES PÚBLICAS.

Trabalho de conclusão de curso apresentado como parte dos requisitos para a obtenção do grau de "Bacharel em Ciências da Computação".
Orientador: Lucas Gonçalves Martins
Coorientador: Prof. Dr. Ricardo Felipe Custódio

Florianópolis

2012

AGRADECIMENTOS

Agradecemos ao Lucas Gonçalves Martins pelo apoio e orientação, assim como aos Profs. Drs. Ricardo Felipe Custódio e Ricardo Pereira e Silva pelo incentivo e dedicação para a realização deste trabalho. Estendemos este agradecimento a todos os colegas do LabSEC.

Agradeço à minha família e aos meus amigos que me apoiaram em toda a minha jornada até concluir este trabalho, dando motivação, incentivo e forças em diversos aspectos. Gustavo Souza Banegas

Agradeço à minha família, que apesar de todos os problemas enfretados, sempre me incentivaram e me apoiaram. Agradeço aos colegas de curso pela amizade cultivada ao longo destes anos todos, aos colegas do LabSEC e ao Prof. Dr. Ricardo Felipe Custódio e Prof. Dr. Ricardo Pereira e Silva, pela ajuda no crescimento moral e intelectual. Aproveito para estender meus agradecimentos ao Prof. Dr. José Mazzucco Junior, que através de suas palavras surgiram forças nos momentos difíceis. Vicente Silveira Inácio

”Os problemas significativos que enfrentamos não podem ser resolvidos no mesmo nível de pensamento em que estávamos quando os criamos.”

Albert Einstein

RESUMO

O Laboratório de Segurança em Computação (LabSEC) da Universidade Federal de Santa Catarina (UFSC) em parceria com Instituto Nacional de Tecnologia da Informação (ITI), desenvolveu uma solução inteiramente nacional que produz e viabiliza toda a cadeia de certificação brasileira, um Sistema Gerenciador de Certificados (SGC), que tem como domínio de atuação a Infraestrutura de Chaves Públicas Brasileira (ICP-Brasil). O projeto João-de-barro, fruto desta parceria, originou um SGC da Autoridade Certificadora (AC) Raiz e um SGC de ACs Intermediárias, Ywapa (raiz, do tupi-guarani) e Ywyra (tronco, do tupi-guarani) respectivamente. Através da experiência obtida com o projeto, o LabSEC vem propor uma nova solução que servirá de base para o desenvolvimento de aplicações de ICP, o *framework* LibPKI. Tendo como referência o SGC, este *framework* tem como domínio a ICP-Brasil (extensível a outras ICPs) e possui a característica de ser uma solução aberta, visando a continuidade e ao melhoramento através da participação da comunidade tecnológica. O uso de *frameworks* no desenvolvimento de aplicações confere ao desenvolvedor uma maior produtividade e qualidade nos artefatos gerados, pois centraliza o esforço em pontos característicos de cada aplicação e estende as partes reusáveis do *framework*. Através da utilização do LibPKI será possível acompanhar o desenvolvimento de um SGC *Online*, e este servirá como modelo de aplicação final no uso do *framework*.

Palavras-chave: ICP, ICP-Brasil, *Framework*, Certificação Digital, Autoridade Certificadora, João-de-barro, SGC.

ABSTRACT

The Computer Security Laboratory (LabSEC) from Federal University of Santa Catarina (UFSC) in partnership with the National Institute of Information Technology (ITI), has developed an entirely national solution that produces and makes the whole chain of Brazilian certification, a Certificate Manager System (CMS), whose field of action is the Brazilian Public Key Infrastructure (PKI-Brazil). The project "João-de-barro", the fruit of this partnership, has led to an CMS for Certificate Authority (CA) Root and Intermediate CA CMS, "Ywapa" (root, from the Tupi-Guarani language) and "Ywyra" (trunk, from the Tupi-Guarani language) respectively. Through the experience obtained from the project, LabSEC proposes a new solution that will provide the basis for the development of applications in PKI, the LibPKI framework. With reference to the CMS, the framework has as its domain the PKI-Brazil (extendable to other PKI) and has the characteristic of being an open solution, aimed at continuing and improving by the community through participation in technology. The use of frameworks in application development, gives the developer greater productivity and quality in the generated artifacts, because the effort is centered on characteristic points of each application and extends the share of reusable frameworks. By using LibPKI, it will be possible to develop a CMS Online, and this will serve as a final application model in the use of framework.

Keywords: PKI, PKI-Brazil, Framework, Digital Certification, Certificate Authority, João-de-Barro, CMS

SUMÁRIO

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

> *"O avanço tecnológico no final dos anos 90 e o uso crescente das vias digitais como suporte para as transações comerciais, bancárias e mesmo de relacionamento entre pessoas e instituições demandou a necessidade da existência de mecanismos que dessem segurança, confidencialidade e, em alguns casos, validade jurídica para essas transações.*
>
> > *"Art. 1º Fica instituída a Infra-Estrutura de Chaves Públicas Brasileira - ICP-Brasil, para garantir a autenticidade, a integridade e a validade jurídica de documentos em forma eletrônica, das aplicações de suporte e das aplicações habilitadas que utilizem certificados digitais, bem como a realização de transações eletrônicas seguras."* (Medida Provisória 2.200-2, de 24 de agosto de 2001)
>
> *A instituição da Infraestrutura de Chaves Públicas Brasileira - ICP-Brasil, por parte do Governo Federal, há dez anos, por meio da Medida Provisória 2.200-2, de 24 de agosto de 2001, veio atender essa necessidade da sociedade brasileira. Prova disso, são os números da ICP. Atualmente, são emitidos um milhão de certificados por ano. As Autoridades Certificadoras credenciadas de primeiro e segundo nível chegam hoje a 45. As Autoridades de Registro saltaram de apenas 12, em 2002, para 389 este ano. O mesmo ocorreu com as instalações técnicas físicas e de ARs lógicas, que chegaram a 885 e 1602, respectivamente, no mês de março deste ano."* (Renato Martini - Presidente do ITI/2011).

Foi com esta medida provisória 2.200-2 que o os cidadãos e instituições brasileiras foram beneficiadas em relação ao uso da internet como um meio alternativo para a disponibilização de serviços com maior agilidade, redução de custos, facilidade de uso e maior segurança. Estes 10 anos de instituição da ICP, comprovam o avanço técnologico do Brasil, com a criação deste mo-

delo de infraestrutura possibilita uma modernização e facilidade de diversos serviços no Brasil.

A ICP oferece suporte a utilização de algoritmos criptográficos que possibilita a um usuário estabelecer uma conexão segura e transferir informações através de uma rede não segura, como a internet, utilizando-se de um par de chaves criptográficos, pública e privada, obtidos através de uma autoridade confiável. Esta infraestrutura fornece ao usuário ou instituição um certificado digital que o identifica.

Uma ICP é constituida pelos seguintes elementos:

- Autoridade Certificadora (AC): Conjunto *hardware-software*. Suas principais funções são: Emissão de certificados, Revogação de certificados e Emissão de LCR.

- Autoridade de Registro (AR): Responsável pela validação dos dados e conexão entre o requerente de um certificado e a AC.

- Lista de Certificados Revogados (LCR): Emitida pela AC, esta lista contém os números seriais dos certificados que não são mais válidos e que não estão expirados.

- Certificados Digitais: Documento digital que contém a chave pública e informações sobre o custodiante da chave privada.

- Sistema Gerenciador de Certificados (SGC): Responsável pela gestão do ciclo de vida das ACs, criação, operação e exclusão.

O desenvolvimento de um SGC para a ICP-Brasil teve início em 2005, e este sistema seguiu as especificações das entidades relacionadas ao projeto João-de-barro. Este projeto do governo brasileiro em parceria com o Laboratório de Segurança em Computação (LabSEC) da UFSC, originou em 2008 uma primeira versão estável do sistema que foi utilizado para a criação da AC Raiz Brasileira. Tecnologia totalmente nacional e auditável.

Atualmente o SGC encontra-se ramificado da seguinte forma: SGC-Ywapa voltada para AC Raiz, SGC-Ywyra para ACs Intermediárias e mais recentemente o SGC-Hawa voltado para entidades finais.

É pensando no futuro da ICP-Brasil que o LabSEC, após ter adquirido vasto conhecimento em certificação digital, propõe uma solução que muda à forma de produzimos SGCs. O presente trabalho diz respeito a elaboração de um arcabouço de *software framework* que está sendo desenvolvido no LabSEC e que viabiliza a produção de SGCs. Os autores deste participam ativamente no processo criativo e gerência do desenvolvimento.

1.2 OBJETIVOS

1.2.1 Geral

Propor um arcabouço de *software* (*Framework*) para a produção de aplicações computacionais no âmbito de Infraestrutura de Chaves Públicas (ICP).

1.2.2 Específicos

- Exemplificar o uso do framework para o desenvolvimento de uma solução completa para autoridades certificadora online;
- Adaptar o modelo dos sistemas gerenciadores de certificados Ywapa e Ywyra, para a estrutura do framework;

1.3 MOTIVAÇÃO E JUSTIFICATIVA

Ao longo dos dez anos de ICP-Brasil, inúmeras aplicações foram desenvolvidas com objetivo de operar sobre este domínio. Porém, de forma descentralizada e engessada, com sistemas distintos implementando funções idênticas, muitas vezes de forma errônea e com definições desencontradas.

Além disso, essas aplicações estão sucetíveis aos efeitos do tempo; algoritmos ob, linguagens de programação e ferramentas, todos caem em desuso, seja por que os algoritmos ficam obsoletos, novas ferramentas mais completas ou pelo simples passar do tempo.

Um arcabouço de *software*, é uma estrutura sólida, porém, que pode ser preparada para receber modificações e atualizações em pontos críticos, sem afetar sua estrutura principal. Dessa forma, um framework surge como a solução para a produção de aplicações que precisam seguir padrões, porém estão sucetíveis a mudanças tecnológicas.

1.4 METODOLOGIA

Apesar dos projetos dos SGCs Ywapa e Ywyra trabalharem com reuso de código e terem envolvido a implementação de uma biblioteca criptográfica, nenhum de seus componentes foram desenvolvidos com características de *framework*. Dessa forma, os autores, como um primeiro passo do projeto da LibPKI, solicitou ao prof. doutor Ricardo Pereira, cujo trabalho de doutorado foi sobre o desenvolvimento de *frameworks*(SILVA, 2000), para que ministrasse um treinamento para toda equipe do projeto. Nesse treinamento, foram exibidas as diferenças entre *software*, biblioteca e *framework*, e o paradigma de programação orientado a componentes.

Após os treinamentos, foram realizadas reuniões para o levantamento dos softwares existentes no domínio da ICP-Brasil e dos seus requisitos. As principais aplicações são os sistemas gerenciadores de autoridades certificadoras raizes, subordinadas e finais, sistemas gerenciadores de autoridades registradoras e módulos públicos. A partir desses requisitos, se deu início a etapa principal do trabalho, a produção da modelagem do framework, que envolveu a criação dos diagramas de casos de uso, atividades, componentes, máquina de estados e classes.

Além dos estudos sobre o desenvolvimento de *framework*, os autores se dedicaram ao estudo do SCRUM(TAKEUCHI; NONAKA, 1986) e desen-

volvimento orientando a testes(BECK, 2003), visando sanar problemas passados com a aplicação de testes automáticos e organização formal da equipe. Deu-se, juntamente com as atividades anteriores, início a configuração de um sistema de integração contínua, para monitoramento aprimorado da implementação da LibPKI, conhecido por Jenkins(CI, 2011).

Através de uma adaptação do *SCRUM*, foram feitas reuniões quinzenais com a equipe, e semanais com a gerência do projeto, nas quais os autores estavam presentes. Nas reuniões iniciais, dividiu-se a equipe em três áreas de desenvolvimento: modelo, interface e persistência. Cada área ficou responsável por definir suas tarefas (ou *sprints*, como é conhecido no *SCRUM*) e prazos, dentro de uma tarefa maior de todo projeto. Após a obtenção dos resultados iniciais, decidiu-se que um protótipo de SGC seria implementado sobre as classes desenvolvidas até então. Esse protótipo tinha o fim de mostrar a viabilidade da modelagem proposta e avaliar a usabilidade do *framework*. Qualquer dificuldade encontrada, resultaria em uma remodelagem do ponto em questão.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 CRIPTOGRAFIA

Analisando a palavra criptografia temos do grego *kryptós* que significa escondido e *gráphien* que significa escrita. A criptografia vem sendo usada desde os tempos mais antigos para transcrever mensagens, tornando-as ilegíveis para quem não conhece o modo de reverter o processo. O processo realizado pelo emissor da mensagem é chamado de cifragem e o processo inverso para tornar os dados legiveis é chamado de decifragem (SINGH, 2001).

Uma das técnicas mais primitivas de criptografia é a transposição de letras, onde cada letra de uma mesagem é substituida por uma outra letra, através de uma tabela prédefinida de substituição. Dessa forma, só será possível decifrar a mensagem sabendo quais letras correspondem as originais. Essa técnica pode ser vista em um cifrador de Cesar, no qual cada letra da mensagem a ser cifrada é substituída pela letra que está algumas posições depois no alfabeto. Por exemplo, defini-se que o segredo do cifrador é substituir cada letra pela próxima letra na sequência do alfabeto, dessa forma, a palavra “criptografia”seria cifrada em “dsjquphsbgjb”, ilegível para quem não possui o segredo. O processo de decifragem ocorreria através da substituição de cada letra da mensagem cifrada pela letra anterior na sequência do alfabeto.

Através dos anos, com o estudo de novos metódos matemáticos aliado ao desenvolvimento de tecnologias foi possivel a melhoria dos metódos de criptografia. Seja através de mecanismos específicos para criptografia como o *Code-o-Graph*,do Capitão da Meia-noite, a máquinas mais sofisticadas como o ENIGMA, do exército alemão na segunda guerra mundial (SINGH, 2001).

Com o invento do computador a criptoanálise de cifras através de transposição de letras ficou simples, pois os criptoanalistas podiam explorar a velocidade e flexibilidade dos computadores programáveis para pesquisar to-

das as chaves possíveis, necessitando assim de uma nova maneira para efetuar a criptografia dos dados (SINGH, 2001). Uma solução adotada foi o conceito de utilização de chave, de forma que para um interceptador entender a mensagem ele precise saber o mecanismo de como foi cifrado, mas também ter posse da chave.

Este mecanismo trabalha com apenas uma chave para cifrar quanto para decifrar é conhecido como criptográfia simétrica. Suas aplicações servem para envio de emails e documentos sigilosos, armazenamento de dados (LUCIANO; PRICHETT, 1987).

O agravante para a comunicação entre duas pessoas fazendo uso da criptografia simétrica, é que ambas devem antes de começar o fluxo necessitam estabelecer uma chave. Para isso é necessário utilizar um canal seguro, o que gera um custo financeiro alto, ou se encontrar pessoalmente, esta última alternativa não é viável caso contrário não seria necessário o uso da criptografia.

2.1.1 Criptografia Assimétrica

Os pesquisadores americanos Whitfield Diffie e Martin Hellman publicaram um artigo em 1976 chamado "*The New Directions in Cryptography*"o qual descrevem um novo método para o problema de troca de chaves. Neste novo modelo, utiliza-se duas chaves, uma chave chamada de pública e outra chamada de privada utilizadas para fazer a troca de uma chave simétrica (HELLMAN, 1976). Este modelo ficou posteriormente, utilizando algoritmos para cifragem como o RSA, conhecido como criptografia assimétrica. Uma vez que seguidas determinadas propriedades matemáticas essas chaves possibilitam que o que for cifrado com chave pública só pode ser decifrado com a chave privada e vice-versa.

Assim como a criptografia convencional, a criptografia assimétrica também tem como funcionalidade a cifra de dados sigilosos, porém suas propriedades vão além da cifragem e decifragem de dados. Suas propriedades

adicionais são a autenticidade e o não-repúdio (SCHNEIER, 1996). Exemplificando as funções de criptografia assimétrica:

- **Sigilo:** Caso Alice queira enviar uma mensagem sigilosa para Bob, basta Alice cifrar a mensagem com a chave pública de Bob. Considerando que Bob é o único que possui a sua chave privada, ele conseguira decifrar a mensagem enviada por Alice.

- **Autenticidade:** Caso Bob deseje saber a autencidade de uma mensagem enviada por Alice. Basta Bob decifrar a mensagem com a chave pública de Alice. Uma vez que Alice é a custodiante da chave privada é a única capaz de enviar a mensagem.

- **Não-repúdio:** Semelhante ao caso de autenticidade, Bob não pode repudiar a autoria de uma mensagem. Se Alice conseguir decifrar a mensagem com a chave pública de Bob, e Bob é o custodiante da chave privada é o único capaz de gerar a mensagem cifrada.

Apesar de possuir um número maior de aplicações e ser mais flexível em relação a troca de chaves, a criptografia assimétrica apresenta uma desvantagem quanto a criptografia simétrica. Operações assimétricas tem um custo computacional mais alto do que operações simétricas (SCHNEIER, 1996). Uma comparação feita na universidade Wuhan, sobre três algoritmos de criptografia, sendo eles: triple-DES, AES e RSA. Eles analisaram o tempo fazendo a cifragem com esses algoritmos, destacando que o triple-DES e AES são algoritmos criptograficos simétricos e RSA é um algoritmo de criptografia assimétrica. Pela comparação feita, o algoritmo triple-DES levou, em média, 0.028 segundos para cifrar um pacote de 4 kb, o AES(256-bit) 0.008917 segundos e o RSA(256-bit) 8.866217 segundos (WANG; HU, 2009).

Para que a criptografia assimétrica possa ser utilizada na prática, é necessária uma forma de garantir a autenticidade das chaves públicas, impedindo que alguém tente divulgar sua chave pública como se fosse de outra

pessoa. Isto pode ser feito por meio de uma Infraestrutura de Chaves Públicas, descrita na seção seguinte.

2.2 INFRAESTRUTURA DE CHAVES PÚBLICAS

Uma infraestrutura de chave públicas tem por função associar uma entidade com seu par de chaves assimétricas. Essa associação costuma ser feita através de um certificado digital, que possui informações da entidades e a chave pública associada a ela. Além disso, é inserido algum tipo de prova, sobre esse certificado, para garantir que a entidade identificada por ele realmente possui a chave privada correspondente a chave pública associada. Na infraestrutura prevista pelo PGP (Pretty Good Privacy), essa prova é constituida pelas assinaturas realizadas por outros certificados, que acreditam que aquele certificado é confiável. Essas trocas de provas acontece de forma semelhante a realação de confiança entre pessoas e cria uma rede de confiança. Apesar de ser um formato aceito pelo comunidade, e muito utilizado para torca de emails, o PGP não possuir valor legal, impedindo que seja utilizado para fins mais importantes.

Outra forma de se organizar uma ICP é através da escolha de um ponto central de confiança, que fica responsável por produzir as provas de que os certificados são válidos. Essa autoridade é chamada de autoridade certificadora raiz (AC). Esse modelo de ICP pode ser adaptado para um modelo hierárquico, onde a autoridade certificadora produz provas para que outras entidades possam, também, produzi-las. Criando, assim, uma cadeia de confiança, que termina na AC raiz, que todos confiam. Por essas características, foi possível atribuir valor legal ao modelo, tornando-o o padrão mundial de certificação digital.

As próximas seções descrevem os principais componentes e estruturas de dados utilizados em uma ICP hierárquica.

2.2.1 Padrão X509

O padrão X509, especificado na RFC-5280, define as estruturas de dados, em ANS.1, pertinentes à construação de certificados digitais e lista de certificados revogados. Esse padrão é utilizado neste trabalho, pois é o mais aceito pelas entidades certificadoras do mundo todo.

2.2.1.1 Certificado Digital

O certificado digital é parte básica de uma ICP. É importante pois contém a chave pública e os dados do custodiante da chave privada correspondente a ela. Fazendo uma analogia, um certificado digital é uma carteira de indentidade (CALLAS, 2007). No padrão X509, os principais campos de um certificado são: número de série, que identifica unicamente o certificado; campo de assunto, que mantém os dados do dono do certificado; o campo de emissor, com as informações referentes ao responsável pela autenticidade dos dados no certificado; data de validade, referente a data de validade do certificado; e a assinatura digital, feita pelo emissor que dá a autenticidade ao certificado. A formalização desses e dos demais campos pode ser encontrada na RFC-5280 (IETF, 1999).

2.2.1.2 Lista de Certificados Revogados

A lista de certificados revogados é utilizda para identificar certificados dentro do prazo de validade que tiveram sua validade revogada por motivos não esperados, como, por exemplo, o comprometimento da chave privada do certificado. Essa lista contém o número serial de cada certificado revogado e informações adicionais, como data e motivo de revogação. Da mesma forma que o certificado digital, essa lista precisa ter uma prova de validade, que pode ser a assinatura de uma terceira parte confiável, como uma autoriade certificadora.

2.2.2 Autoridade Certificadora

A autoridade certificadora é a entidade, dentro da ICP, responsável por gerar provas de validade para os certificados digitais e listas de certificados revogados. Essas provas são assinaturas digitais, feitas com a chave privada da autoridade certificadora, sobre as informações contidas no certificado. Essa, e outras funções da AC, estão listadas a seguir:

- **Emissão de certificados:** A AC pode emitir certificados para entidades finais e outras ACs. O processo de emissão de certificado consiste na construção de um certificado digital de acordo com uma requisição de certificado feita pelo interessado. Essa requisição está sujeita a alterações, conforma a vontade da AC. Durante o processo, a autoridade certificadora precisa incluir o nome do emissor e o número serial do certificado, além de extensões referentes ao tipo do certificado e o ponto de distribuição da lista de certificados revogados, onde esse certificado pode vir a ser inserido. Uma vez gerado o certificado, a AC o assina com sua chave privada. Ao emitir um certificado, ela cria uma prova de que o titular (entidade nomeada no certificado) possui a chave privada que corresponde a chave pública contida no certificado.

- **Revogação de certificados:** Um certificado pode ser revogado pela autoridade certificadora que o emitiu. A revogação é um processo simples, onde a AC deve criar um registro do número serial do certificado revogado, junto com a data de revogação e informações adicionais, se existirem. Uma vez revogado, o certificado será inserido na próxima lista de certificados revogados emitida pela AC.

- **Emissão da Lista de Certificados Revogados:** Conforme definido no glossário ICP-Brasil, uma lista de certificados revogados é uma lista assinada digitalmente por uma Autoridade Certificadora, publicada periódicamente, contendo certificados que foram revogados por

> determinados motivos. A lista, geralmente, indica o nome de quem a emite, a data de emissão e a data da próxima emissão programada, além dos números de série dos certificados revogados e suas datas de revogação (ICP-BRASIL, 2010).

Segundo Housley e Polk (POLK, 2001), em uma ICP hierárquica tradicional, existem múltiplas ACs que fornecem serviços de ICP. Estas entidades estão conectadas entre si em um modelo Superior-Subordinado. A AC Raiz é superior a todas as outras ACs (Autoridade Certificadora Intermediária ou Normativa e Autoridade Certificadora Final), e tudo que estiver abaixo dela é considerado confiável, pois todas as entidades confiam em uma mesma AC Raiz central. As AC's Intermidiárias e Finais, são consideradas AC's Subordinadas, pois tem seus certificados emitidos por outras AC's e serão assim tratadas pelos autores neste trabalho.

2.2.2.1 Autoridade Certificadora Raiz

Uma autoridade certificadora raiz é o ponto central de confiança em uma ICP hierárquica. Ela é identificada por um certificado digital, que possui uma prova de validade (assinatura) gerada por ela mesmo. Por ser o ponto de confiança, se uma autoriade certificadora raiz for comprometida, toda cadeia de certificação, abaixo dela, também é comprometida. Por essa razão, uma AC raiz precisa ser gerenciada por uma entidade confiável, como o governo de um país, que garanta o uso correto do par de chaves da AC.

O certificado da autoridade certificadora raiz costuma possuir um prazo de validade grande e utilizar algoritmos criptográficos mais poderosos para a geração da sua prova de validade. O ambiente de gerenciamento é offline e restrito a poucas pessoas, é comum o uso de dispositivos especialiazado no gerenciamento de chaves para guardar e registrar os usos da chave privada da AC. Por não emitir tantos certificados, a lista de certificados revogados possui intervalos longos de emissão.

2.2.2.2 Autoridade Certificadora Intermediária

Diferente de uma AC Raiz, que tem seu certificado autoassinado, as ACs Intermediárias tem seus certificados assinados por uma AC superior, sendo ela raiz ou outra ac subordinada (POLK, 2001). Essas ACs criam subcadeias de certificação, reduzindo as responsabilidades da AC Raiz. Uma AC é considerada intermediária enquanto ela puder emitir certificados para outras autoridades. Na ICP-Brasil, essas autoridades são conhecidas por ACs Normativas.

Da mesma forma que AC raiz, ACs intermediárias ainda possuem certificados com prazos de validade longos. Seu gerenciamento deve ser restrito da mesma forma que da AC Raiz, offline e com o uso de dispositivos criptográficos para o gerenciamento de suas chaves. As LCRs possuem um intervalo menor de emissão, dependendo do nível em que a AC se encontra na hierarquia.

2.2.2.3 Autoridade Certificadora Final

Pode descender diretamente de uma AC Raiz ou de uma AC Intermediária, seu papel na estrutura da ICP é emitir certificados para usuários finais. Assim, abaixo dela não pode existir nenhuma outra AC. Suas funções são emitir, revogar e gerar lista de certificados revogados para usuários finais (POLK, 2001).

A Autoridade Certificadora Final (ou Online) possuem uma grande demanda de emissão de certificados para usuários e por consequência, a emissão de LCR é constante, gerando assim um grande fluxo de dados nessa AC. Como a emissão de seus certificados são destinados a usuários finais, a validade destes é reduzido em comparação aos certificados emitidos por uma AC intermediária ou AC raiz. Apesar de trabalhar em ambiente online, ACs finais também precisar usar dispositivos criptográficos no gerenciamento de suas chaves.

2.2.3 Módulo de Segurança Criptográfico

A chave privada de um AC é um dos pontos críticos de uma ICP, pois seu comprometimento invalida todas suas assinaturas, ou seja, os certificados emitidos pela AC. Por esse motivo, autoridades certificadora precisam utilizar dispositivos criptograficos para gerenciar suas chaves. Esses dispositivos são o conjunto de software e hardware que garantem geração, armazenamento e uso seguro de chaves criptográficas. A comunicação com esses disposivos ocorre através de protocolos bem definidos, como a Engine OpenSSL e a API PKCS#11.

2.2.4 Autoridade de Registro

Devido a limitações físicas e geográficas, autoridades certificadora fazem uso de autoridades registradoras (ARs) para identificar e autenticar os requerentes de certificados. Assim, a AR deve garantir à AC que as requisições de certificados possuem um vínculo válido, e legal, entre pessoa e chave pública (POLK, 2001).

Segundo Housley e Polk (POLK, 2001), cada AC mantém uma lista de ARs que considera confiáveis. Cada AR é conhecida pela AC pelo seu nome e sua chave pública. Por verificar a assinatura de uma AR na mensagem, a AC tem certeza que quem forneceu a informação foi uma AR confiável, e sendo assim a AR deve fornecer proteção adequada a sua chave privada, usando sempre um MSC(Módulo de Segurança Cripotográfico) validado pela FIPS 140.

Através de uma conexão segura, a AR confirma os dados do requerente, através da requisição que o mesmo efetuou para a AC, e então salva o certificado do requerente em hardware criptográficos adequado (e.g, Token, SmartCard) (POLK, 2001).

2.2.5 Módulo Público

O Módulo Público vem dentro do escopo de ICP para criar uma abstração da RFC2511 (**??**) que referência a geração de requisição de certificado. A abstração em cima da RFC serve para criar uma interface convidativa para a geração de chaves e requisições de certificado. Essa interface é disponibilizada por ARs aos seus usuários, através de aplicações online.

A principal vantagem do uso do Módulo Público é a utilização de uma interface amigável para o preenchimento dos requisitos para gerar o par de chaves, seja em disco ou em um hardware criptográfico, ou para enviar a requisição de um certificado para uma AR.

2.3 FRAMEWORK

Uma definição para *framework* pode ser descrita da seguinta maneira, uma estrutura de classes que corresponde a uma implementação incompleta para um determinado domínio de aplicações. Um *framework* se destina a atender diferentes aplicações de um mesmo domínio (SILVA, 2000). Porém, esta aplicação não precisa ser uma aplicação final. Sendo assim, pode-se descrever um *framework* como um conjunto de classes abstratas ou concretas que resolvem um determinado domínio de problemas.

Wirfs-Brock define *framework* como "Um esqueleto de implementação de uma aplicação ou de um subsistema de aplicação, em um domínio de problema particular. É composto por classes abstratas e concretas e provê um modelo de interação ou colaboração entre as instâncias de classes definidas pelo framework".

A definição de Booch et al.(2006) foi usada na classificação do framework em desenvolvimento:

> *"Um framework é um padrão de arquitetura que fornece um template extensível para aplicações dentro de um domínio. Você pode pensar em um conjunto de mecanismos que trabalham jun-*

> *tos para resolver um problema básico de um domínio comum. Ao especificar um framework você especifica o esqueleto da arquitetura, juntamente com os conectores, guias, botões e indicadores que você expõe aos usuários que desejam adaptar esse framework ao seu próprio contexto.*" (BOOCH; RUMBAUGH; JACOBSON, 2006)

Estas definições expostas podem gerar algumas dúvidas pois assemelham-se muito com o funcionamento de uma biblioteca de software. Ambas apresentam classes "incompletas", funcionam sobre determinados domínios e necessitam da interação com o usuário para definição da semântica do sistema. Porém a grande diferença que define ambos é a que segue:

"A diferença entre *framework* e a reutilização de classes de uma biblioteca, é que em uma biblioteca cabe ao desenvolvedor estabelecer suas interligações e em um *framework* as classes já está inter-relacionada." (SILVA, 2000)

O objetivo principal (ou vantagem) de um *framework*, é prover uma redução significativa no tempo de desenvolvimento de sistemas, e o mais importante, pontencializar o reuso de *software*. Porém, com o planejamento há um grande desafio a ser vencido, a complexidade de desenvolvimento e de uso.

"Em termos práticos, dotar um framework de generalidade e flexibilidade requer uma cuidadosa identificação das partes que devem ser mantidas flexíveis e a seleção de soluções de projetos de modo a produzir uma arquitetura bem estruturada. Isto passa pela observação de princípios de projeto orientado a objetos, como o uso de herança para a reutilização de interfaces (ao invés do uso de herança para reutilização de código); reutilização de código através de composição de objetos; preocupação em promover polimorfismo, na definição de classes e métodos, etc. No contexto da abordagem *frameworks* o uso adequado de herança implica na concentração das generalidades do domínio em classes abstratas, no top da hierarquia de classes. Isto promove o uso adequado de herança, pois a principal finalidade dessas classes abstratas é definiar as interfaces a serem herdadas pelas classes concretas

das aplicações." (SILVA, 2009)

2.3.1 Arquitetura de um *Framework*

2.3.1.1 *Frozen spots*

Frozen spots definem a arquitetura global de um sistema de software – seus componentes básicos e os relacionamentos entre eles. Eles permanacem imutáveis em qualquer instanciação do *framework*. (UCHÔA, 1999)

2.3.1.2 *Hot spots*

Hot spots representam aquelas partes do framework que são específicas para cada sistema de software. *Hot spots* são projetados para serem genéricos – eles podem ser adaptados para as necessidades da aplicação em em desenvolvimento. Quando se cria um sistema de software concreto, usando um framework, seus *hot spots* são preenchidos de acordo as necessidades e requisitos específicos dos sistema (UCHÔA, 1999).

2.3.2 Classificação

Os *frameworks* podem ser classificados em caixa-branca e caixa-preta (FAYAD; SCHMIDT; JOHNSON, 1999). Ainda deve ser considerado uma terceira classificação, a caixa-cinza, que seria uma mistura das outras duas.

Os *frameworks* caixa-branca baseiam-se nos mecanismos de herança e ligação dinâmica (*dynamic binding*) presentes em orientação a objetos. Os recursos existentes em um framework caixa-branca são reutilizados e estendidos a partir de: herança de classes do framework e sobrecarga (*overriding*) de métodos "*hook*" pré-definidos. Métodos "*hook*" são definidos em interfaces ou classes abstratas e devem necessariamente ser implementados por uma aplicação. Os padrões de projeto (design patterns) utilizados são o *Command*,

o *Observer* e o *Template Method* (GAMMA, 1995).

Frameworks caixa-preta são baseados em componentes de software. A extensão da arquitetura é feita a partir de interfaces definidas para componentes. Os recursos existentes são reutilizados e estendidos por meio de: definição de um componente adequado a uma interface específica e integração de componentes em um framework que utiliza padrões de projeto como o *Strategy* (GAMMA, 1995).

Quanto à utilização, (FAYAD; SCHMIDT; JOHNSON, 1999) propõe que os frameworks sejam classificados em:

- Framework de Infraestrutura de Sistema: tratam de questões de projeto como sistemas operacionais, comunicação, interfaces gráficas e linguagens de programação.

- Framework de Integração de Middleware: responsáveis por integrar aplicações distribuídas e componentes em uma mesma arquitetura.

- Framework de Aplicação: tratam de questões de projeto de domínios de aplicação, como telecomunicações, finanças, produção e educação.

O objeto deste trabalho é classificado como um *Framework* de Aplicação.

2.4 COMPONENTES

Em 1976 DeRemer propõe um paradigma de desenvolvimento diferentes dos demais, propõem que o software pode ser desenvolvido em um conjunto de módulos produzidos de forma separada e que depois de terminados podem ser interligados (SILVA, 2000).

A abordagem de desenvolvimento orientado a componentes é semelhante ao trabalho proposto por DeRemer (SILVA, 2000). ”O que torna algo um componente não é uma aplicação específica e nem uma tecnologia de implementação específica. Assim, qualquer dispositivo de software pode ser considerado um componente, desde que possua uma interface definida. Esta

interface deve ser uma coleção de pontos de acesso a serviços, cada um com uma semântica estabelecidade"(SILVA, 2000).

Ainda no escopo de desenvolvimento orientado a componentes, pode-se fazer uma analogia em que cada componente é um módulo que funciona separado, e que funciona também se for interligado com outros módulos. Esta interconexão dos componentes deve ser estabelecida atráves do uso de interfaces comuns, para que todos possam ser interligados. É importante destacar que um componente pode ser integrante interno de outro componente por exemplo, uma engrenagem pode ser vista como um componente, ela está contida dentro de um motor e este é outro componente.

Pode-se descrever um componente através da descrição de sua interface, porém esta é uma visão muito limitada, pois pode não ser possível chegar a um entendimento completo das suas funcionalidades e comunicações.

Segundo Silva, a descrição de um componente deve abranger três aspectos distintos:

- Estrutural;
- Comportamental;
- Funcional.

"A descrição estrutural corresponde à relação das assinaturas de métodos da inteface de componente. A descrição comportamental também se atém à interface de componente e especifica restrições na ordem de invocação de métodos. A descrição funcional vai além da interface de componente e visa descrever o que o componente faz, pois, pode não ser possível compreender o que os métodos fazem apenas conhecendo suas assinaturas e restrições de ordem de invocação." (SILVA, 2009)

Um componente pode ser analisado como uma composição de artefatos, tais como portos, interfaces e máquinas de estados que ditam como este se comunica com o meio externo, oferecendo ou requisitando serviços e definindo o fluxo destas comunicações.

2.4.1 Porto

"Representa a fronteira entre uma classe, componente ou pacote e seu meio externo. Corresponde a um ponto de conexão entre o elemento que o possui e seu exterior, bem como uma ligação entre a fronteira e sua estrutura interna. A um porto podem ser associadas uma ou mais interfaces (UML) que especificam métodos acessíveis naquele ponto de conexão." (SILVA, 2009)

O porto é a porta para o mundo externo, onde estão associadas interfaces requeridas pelo componente para fazer uso de recursos externos e nele também estão associadas interfaces oferecidas pelo componente, na qual elementos externos utilizam serviços disponibilizados pelo componente.

2.4.2 Interface

Segundo Silva, interface é *"uma relação de assinaturas de métodos a ser implementada por uma classe à qual a interface esteja associada, através de uma relação de realização. Uma interface também pode conter dados, na forma de constantes."* (SILVA, 2007, p.205)

A interface contratualmente especificada, oferece uma gama de serviços disponíveis à relação do componente com a classe associada. Estes serviços são oferecidos na forma de métodos e dados constantes associados em uma relação de realização.

2.4.3 Máquina de Estados

A máquina de estados está associada a modelagem comportamental de interface de componente, que estabelece restrições na ordem de execução de métodos fornecidos e requeridos pelo componente.

"A idéia básica é que cada estado represente um conjunto de métodos fornecidos ou requeridos que podem ser invocados em um determinado instante. Cada transição que sai de um estado representa a execução de um

método fornecido ou requerido que pode deixar o componente no mesmo estado (isto é, o mesmo conjunto de métodos pode ser executado) ou levar a outro estado, caracterizado por outro conjunto de métodos que pode ser executado." (SILVA, 2009)

2.5 ABSTRACT SYNTAX NOTATION ONE

O ITU-T define o ASN.1 da seguinte forma:

> *"A notação de sintaxe abstrata número 1 é um padrão que define o formalismo para a especificação de tipos de dados abstratos."*

O ASN.1 é uma notação que define os tipos de dados simples e complexos, e os determinados valores que estes podem assumir. Define apenas a estrutura, e não os valores dos dados. Ela define as regras e codificações que os dados precisam ser submetidos para serem utilizados nos protocolos de comunicação, independentemente da linguagem. Fazendo o uso de compiladores especiais, é possível tomar como entrada especificações ASN.1 e gerar código em linguagens de programação qualquer, e obter de maneira equivalente as estruturas de dados.

ASN.1 foi padronizado em 1984 pelo CCITT (*International Telegraph and Telephone Consultative Committee*) atualmente chamado de ITU-T (*International Telecommunication Union - Telecommunication Standardization Sector*) sob o nome de "X.409 *Recommendation*". Pouco tempo depois, a ISO (*International Organization for Standardization*) decidiu adotar a notação e a separou em dois documentos: a sintaxe abstrata (ASN.1) e as regras de codificação (BER).

> *"Abstract Syntax Notation One (ASN.1) é uma notação que é usada para descrever mensagens que serão trocadas entre programas de aplicações comunicantes. Ela provê descrições de alto nível de mensagens que livram o projetista de protocolos de ter que focar nos bits e bytes do "layout"das mensagens. Inicialmente usada para descrever mensagens de email dentro dos*

```
Certificate  ::=  SEQUENCE  {
     tbsCertificate       TBSCertificate,
     signatureAlgorithm   AlgorithmIdentifier,
     signatureValue       BIT STRING  }

TBSCertificate  ::=  SEQUENCE  {
     version         [0]  EXPLICIT Version DEFAULT v1,
     serialNumber         CertificateSerialNumber,
     signature            AlgorithmIdentifier,
     issuer               Name,
     validity             Validity,
     subject              Name,
     subjectPublicKeyInfo SubjectPublicKeyInfo,
     issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                          -- If present, version MUST be v2 or v3
```

Figura 1 – "Exemplo de especificação em ASN.1. Trecho dos campos básicos do certificado X.509 - fonte: RFC5280"

> *protocolos do modelo Interconexão de Sistemas Abertos (OSI), ASN.1 foi adotado para o uso em uma ampla variedade de aplicações, tais como gerenciamento de rede, email seguro, telefonia celular, controle de tráfego aéreo, voz e vídeo sobre a internet."* (DUBUISSON, 2001)

Esta notação surgiu com o intuito de formalizar as comunicações entre sistemas distribuídos, que requisitam padrões semânticos nas transferências de suas informações. A notação passa por constantes melhorias e atualizações, em 1988 passou a suportar certificados digitais X.509. Amplamente utilizada em diversos tipos de aplicações computacionais e distribuído em bilhões de computadores e dispositivos com sistemas embarcados.

3 APLICAÇÕES EM INFRAESTRUTURA DE CHAVES PÚBLICAS

Cada seção deste capítulo descreve as funcionalidade das principais aplicações que integram uma ICP. O objetivo desse levantamento é ter um conjunto inicial das aplicações e funções que a LibPKI deve suprir. A especificação detalhada de cada aplicação permite realizar uma análise comparativa entre elas, para que sejam encontradas estruturas, funções e protocolos semelhantes. O resultado dessa análise permite definir os pontos que devem ser fixados pelo framework e os que devem ser customizáveis pelo usuário, facilitando, assim, a modelagem do framework.

A descrição das funções foi feita com base nos softwares já existentes, Ywapa e Ywyra, e em estudos realizados sobre as entidades que operam na ICP.

3.1 SISTEMA GERENCIADOR DE CERTIFICADOS

Sistema gerenciadores de certificados (SGC) são responsáveis pelo ciclo de vida de autoridades certificadoras. Suas principais funções são: criar autoridade certificadora, emitir certificado, revogar certificado e emitir lista de certificados revogados. Porém, além dessas funções, o SGC deve estar preparado para criação e recuperação de backups, geração e exportação de logs auditáveis, criação e uso de modelos de certificados, geração e exportação de relatórios operacionais, e controle de acesso ao sistema, conforme especificado pelo Manual de Condutas Técnicas disponibilizado no site do ITI (ICP-BRASIL, 2010).

Nas próximas seções serão descritas as principais funções do SGC, e como elas se diferenciam entre ACs raizes, intermediárias e finais. Também serão dados exemplos, da execução dessas funções, baseados nos sistemas gerenciadores de certificados Ywapa e Ywyra.

3.1.1 Criação de Autoridade Certificadora

A criação de uma autoridade certificadora consiste na criação do seu par de chaves e na geração de seu certificado digital. Para autoridades certificadoras raizes, a criação do certificado consiste no preenchimento dos campos do certificado e na realização da assinatura do mesmo, com a própria chave da AC. Autoridades subordinadas (intermediárias e finais) possuem um processo diferente. Como seus certificados não são autoassinados, elas precisam gerar uma requisição de certificado, para que, então, outra AC emita o certificado para ela. Apesar das diferenças, boa parte do processo é semelhante para ambos tipos de AC.

A seguir, as etapas da criação de AC são detalhadas. A ordem em que estão listadas não representam uma ordem de execução, porém, algumas delas possuem relações de dependência.

- **Geração do Par de Chaves:** A geração do par de chaves de uma AC deve ser feita, preferencialmente, em um módulo de segurança criptográfico. Porém, cada MSC possui seus protocolos de geração de chaves, o que dificulta a implementação dessa função no SGC. No Ywapa e Ywyra, quando é utilizado um MSC, a criação da chave acontece fora do sistema, mas não deixa de fazer parte da creimônia de criação da AC. Caso a AC não use MSC, o SGC deve garantir o sigilo e integridade da chave, armazenando-a cifrada e com controle de acesso.

- **Montagem do Certificado:** A montagem do certificado de uma AC consiste no preenchimento de um formulário que representa os campos de um certificado. Esse formulário deve ser adaptado conforme o tipo de AC que esá sendo criada. Autoridades certificadoras raizes devem preenche-lo com a visão de emissor de certificado, podendo editar todos os campos existentes, para que seja gerada uma estrutura de certificado, conforme especificada na RFC-5280. Autoridades certificadoras subordinadas possuem uma visão mais restritia, podendo editar apenas

campos da responsabilidade do requerente, para que seja gerada uma estrutura de requisição de certificado, conforme o padrão PKCS#10.

- **Carregamento da chave:** O carregamento das chaves ocorre de formas diferentes, dependendo de como o par de chaves está sendo armazenado. Quando a AC faz uso de um MSC, as chaves devem ser carregadas a partir das configurações, específicas, da interface de comunicação que o MSC utiliza, como, por exemplo, Engine OpenSSL e API PKCS#11. Em alguns casos, será necessário o uso de um software externo, para liberação de uso da chave no MSC. Mais detalhes sobre o processo de carregamento de chaves são dados na seção 3.1.7;

- **Realização de Assinatura:** Após as três etapas anteriores, o SGC está pronto para realizar a assinatura sobre as informações do certificado. Para isso, é preciso definir qual algoritmo de hash deve ser utilizado na assinatura, para então assinar o certificado, ou requisição de certificado, dependendo do tipo da AC que esta sendo criada. Para autoridades certificadoras raízes, essa é a última etapa, pois, ao fim dela, seus certificados estarão emitidos. Autoridades subrodinadas devem seguir os próximos dois passos, para finalizar o processo de criação de AC.

- **Emissão de certificado da AC:** Uma autoridade certificadora subordinada deve exportar sua requisição de certificado e enviá-la para uma outra autoridade certificadora, para que essa emita seu certificado. Após a exportação do certificado, todo processo de emissão está fora do escopo do SGC, porém faz parte da cerimônia de criação da AC.

- **Importação de Certificado:** A importação de certificado acontece após a etapa de emissão do certificado da ac subordinada. Esse processo pode ser feito através da importação de um arquivo que contém o certificado, que deve ter suas informações e chave pública verificados, garantindo que o certificado representa a AC em questão.

Durante o processo de criação, é recomendado que se defina os responsáveis pela operação da AC. Na seção 3.1.8, são dadas sugestões de como gerenciar o controle de acesso às funções da AC.

3.1.2 Emissão de Certificados

A emissão de certificado é tratada, no SGC, como o processo de assinar, com a chave da AC, uma estrutura de certificado contendo informações de um requerente. Autoridades certificadoras offline fazem a emissão de certificado de forma manual, enquanto autoridades online possuem protocolos automáticos de emissão de certificado.

Para se emitir um certificado é precisso que, anteriormente, uma requisição de certificado tenha sido gerada, contendo as informações do requerente, sua chave pública e provas de autenticidade dessas informações. O Ywapa e Ywyra, que operam ACs offline, fazem uso do padrão PKCS#10, para requisições de certificado. Esse formato utiliza as mesmas estruturas definidas na RFC-5280, para construir uma estrutura de dados contendo os dados do requerente de certificado, junto com uma assinatura, realizada com a chave privada do mesmo. A validação das informações, nesse caso, ocorre de forma manual, pois os SGCs, Ywapa e Ywyra, não trabalham com autoridades de registro.

No caso de autoridades certificadoras online, que costumam trabalhar com autoridades de registro, é preciso utilizar formatos diferentes de requisição de certificado, pois é preciso incluir informações adicionais, como a assinatura da AR, que é a prova da validade dos dados da requisição. Protocolos como o CMC e CMP propõem estruturas de dados para esses tipos de requisições.

Os passos para emissão de certificado, conforme acontecem no Ywapa e Ywyra, estão descritos a seguir:

- **Importação da Requisição de Certificado:** A importação da requisição

de certificado acontece através de um arquivo contendo a estrutura da requisição no formato PKCS#10. Nesta etapa são feitas verificações da codificação e integridade do arquivo.

- **Validação da Requisição:** Essa etapa inclui todas verificações feitas sobre o conteúdo da requisição. Uma dessas verificações é sobre a prova de posse da chave privada, que, no caso de requisições no formato PKCS#10, é feita através da verificação da assinatura da requisição com a chave pública contida na própria requisição. As verificações de conteúdo deve ser feita manualmente pelo operador da AC.

- **Montagem do Certificado:** A autoridade certificadora emissora pode alterar qualquer campo que irá fazer parte do certificado, independente do conteúdo da requisição importada. A única informação que não deve ser alterada é a chave pública da requisição, pois ela identifica a chave privada do requerente. Os SGCs Ywapa e Ywyra permitem a edição através de um wizard de construção de certificado, pré-preenchido com as informações da requisição importada. Nesse momento é possível fazer o uso de modelos de certificados, que possuem uma estrutura padrão, com as informações que devem ser inseridas no certificado, para sua emissão.

- **Realização da Assinatura:** Após as três etapas anteriores, o SGC está pronto para realizar a assinatura sobre as informações do certificado. Para isso, é preciso definir qual algoritmo de hash deve ser utilizado na assinatura, para então assinar o certificado.

- **Exportação do Certificado:** Após a assinatura do certificado, deve ser possível exporta-lo, para publicação. Essa exportação pode ser feita, por exemplo, para um arquivo, contendo a estrutura X.509 no certificado.

3.1.3 Revogação de Certificados

O SGC deve permitir a revogação de certificados emitidos pela AC em operação. A revogação envolve, pelo menos, três informações relevantes: o número serial do certificado, a data da revogação e informações sobre o motivo da revogação. No caso de autoridades certificados online, a revogação de um certificado acontece a partir de uma requisição de revogação de certifcado, enviada por uma das autoridades registradoras que a AC confia. Nos SGCs Ywapa e Ywra, que operam auotirades offline, a revogação de certificados acontece da seguinte forma:

- **Seleção do Certificado:** O SGC apresenta uma lista de certificados emitidos, e não revogados, pela AC, para que o operador escolha o certificado que deseja revogar. Essa escolha define o número serial do certificado a ser revogado.

- **Definição do Motivo da Revogação:** Após a escolha do certificado, o SGC solicita o código do motivo da revogação, conforme previsto na RFC-5280, e um texto detalhando esse motivo, conforme as regras de negócio dos SGCs Ywapa e Ywyra.

- **Revogação:** É feita a persistência do número serial do certificado, como revogado, junto com as informações do motivo da revogação e a data em que o certificado foi revogado (a data no momento da revogação).

3.1.4 Emissão de Lista de Certificados Revogados

A emissão de LCR, para o SGC, consiste na construção de uma estrutura que contêm: a lista dos números seriais dos certificados que foram revogados, a data de emissão da LCR, a data de atualização, que será emitida a próxima LCR, e a assinatura, realizada sobre a estrutura, com a chave da

AC. É recomendado que a emissão de LCR aconteça sempre na data prevista pela última LCR emitida, para que não exista mais de uma LCR válida ao mesmo tempo.

Autoridades certificadoras online emitem listas de certificados revogados em um curto intervalo de tempo, por esse motivo, os SGCs que operam essas ACs, devem prover ferramentas automáticas de emissão de LCR, evitando lacunas ou sobreposições de LCRs. Porém, exitem casos onde um certificado deve ser inserido em uma LCR imediatamente, para isso, o SGC deve permitir a emissão manual de LCRs. A AC pode permitir, também, que autoridades registradoras solicitem a emissão de listas de certificados revogados. Para isso, podem ser usadas as estruturas definidas pelos protocolos CMC e CMP, para requisição de LCR.

Autoridades certificadoras offline, por sua vez, emitem LCRs em longos intervalos de tempo. Além disso, os SGCs que operam esses tipos de ACs costumam ficar desligados, impedindo que a emissão de LCRs sejam automáticas. Portanto, para ACs offline, a emissão de LCR é feita de forma manual, conforme descrito a seguir:

- **Definição do Intervalo de Emissão:** O intervalo de emissão de LCR, no Ywapa e Ywyra, é definido em dias. Ele é usado para calcular a data de atualização da LCR, somando seu valor com a data de emissão da LCR.

- **Montagem da LCR:** A montagem da LCR consiste na construção de uma estrutura de dados em ASN.1, definida na RFC-5280, contendo todos certificados revogados, não expirados, da AC. Durante o processo são inseridas informações do emissor da LCR, bem como as datas de emissão e atualização da LCR.

- **Realização da Assinatura:** Após as duas etapas anteriores, o SGC está pronto para realizar a assinatura sobre as informações do certificado. Para isso, é preciso definir qual algoritmo de hash deve ser utilizado na assinatura, para então assinar a LCR.

- **Exportação de LCR:** Após a assinatura da LCR, deve ser possível exporta-la, para publicação. Essa exportação pode ser feita, por exemplo, para um arquivo, contendo a estrutura ASN.1 da LCR.

3.1.5 Geração e Recuperação de *Backup*

Uma das funções que é desejável a uma AC de acordo com o MCT(ICP-BRASIL, 2010), no entando não essêncial é a geração de um *backup* dos dados da AC. Os dados salvos devem conter todas as operações que aconteceram no SGC, por consequência todas as ações que ocorreram nas AC que são gerenciadas.

Operações que incluem a emissão e revogação de certificados, emissão de LCR, entre outras e todas as ações que ocorreram, como por exemplo a ativação e desativação da AC. Bem como a questão da utilização do *software* SGC incluindo os momentos em que foi aberto e fechado, isso será importante para o próximo item da geração de *logs*, todos estes eventos devem ser salvos em um base de dados, que posteriormente seja possível a efetuação de um *backup*.

Não existe um formato específico para o salvamento dos dados, apenas que os dados sejam salvos de forma que apenas o grupo de administradores consiga recuperar. A seguir será apresentado passos, de uma forma genérica, para a o salvamento dos dados:

- **Selecionar algoritmo:** Deverá ser apresentado ao usuário que está efetuando o *backup* qual algoritmo criptográfico que vai ser utilizado para cifrar o arquivo contendo o salvamento dos dados.

- **Salvamento dos dados:** Através de uma forma eficaz e eficiente, deve-se salvar os dados em um arquivo pode ser utilizando o *dump* de um SGBD(Sistema Gerenciador de Base de dados). O importante nesta etapa é a geração de um arquivo que possa ser cifrado.

- **Cifragem do *backup*:** Após as duas etapas anteriores, o SGC deve uti-

lizar o algoritmo selecionado e o arquivo gerado para gerar um arquivo cifrado, que apenas os administradores do SGC possam recuperar.

- **Compressão do arquivo cifrado:** Este passo pode ser considerado supérfluo, pois a compreensão dos dados serve para diminuir e/ou juntar, caso exista mais de um arquivo cifrado, em um arquivo que ocupe menos espaço para salvamento.

Após a geração do *backup* uma funcionabilidade que é desejável é a recuperação do mesmo. Isto é importante para caso a base de dados do SGC seja corrompida ou por alguma razão torne-se inválida.

Para efetuar a recuperação do *backup* da base de dados é necessário seguir alguns passos que serão apresentados a seguir:

- **Descrompressão do arquivo:** Caso foi implementado que na última etapa do processo de geração de *backup* ele irá realizar a compressão do arquivo, então é mais que necessário que o primeiro passo seja a descompressão do mesmo.

- **Autenticação dos administradores:** Como o arquivo está cifrado, para efetuar a decifragem deste arquivo é necessário que os administradores se autentiquem no SGC. Somente com a autenticação dos administradores é possivel decifrar e visualizar o conteúdo do(s) arquivo(s).

- **Recuperação do *backup*:** Após as duas etapas anteriores, o SGC terá condições de ler o(s) arquivo(s) e lendo-os será possivel efetuar a recuperação deste *backup*.

3.1.6 Geração de Logs Auditáveis

Os logs auxiliam na detecção de anormalidades no sistemas, desde ataques/intrusões a anomalias em funcionalidades. A importância de registrar os eventos serve para o acompanhamento do histórico de ações de uma AC ou SGC em uma auditoria, de maneira simples e eficaz. A geração destas

informações deve ser requisitada pelos níveis de controle de acesso mais altos, como administradores do sistema, para garantir a integridade e preservação dos dados.

- **Políticas de log:** Definição das informações contidas em um log, e.g: Tipo do evento, AC responsável pela ação (caso a ação seja efetuada por uma AC), hora do evento, data do evento e se o evento foi bem sucedido ou não.
- **Registro no log:** Os diversos fluxos de execução do sistema, e.g: Emissão de certificado, são registrados no log seguindo as políticas defindas.
- **Armazenamento de log:** Os códigos de log são armazenados codificados e estes liberados por meio de autenticação pelo nível de acesso.
- **Análise e Monitoramento de log:** Os logs são visualizáveis através de buscas, com ou sem uso de filtros, para controle administrativo.
- **Exportação de log:** Geração de um arquivo de log, podendo conter as informações filtradas na busca ou o histórico completo do sistema.

3.1.7 Suporte a Módulo de Segurança Criptográfico

A utilização de módulos de segurança criptográficos (hardware ou software) se faz necessário em um SGC para que este garanta uma segurança adequada ao gerenciamento de chaves criptográficas. Esse suporte ao módulo deve fornecer um meio de comunicação para a execução de funcionalidades, tais como: geração, carregamento das chaves e uso de algoritmos criptográficos, através de interfaces que atendam todas as diferentes formas de implementação em MSC's, e até o uso de software complementares. Um SGC fornece o gerenciamento de MSC por meio destas funções:

- **Cadastrar MSC:** O cadastramento de um MSC, baseado em engine OpenSSL, se dá através da definição de um identificador do MSC, do

identificador da engine, escolha da engine em arquivo, endereço IP e seus comandos.

- **Gerenciar MSC:**

  - **Atualizar:** Alterar as configurações de um MSC cadastrado no sistema.

  - **Remover:** A remoção de um MSC só é possível quando não há AC's vinculadas.

- **Utilizar MSC:** Quando da criação de uma AC, o SGC fornece a opção de utilizar uma chave que encontra-se armazenada em um MSC cadastrado e que fora previamente gerada.

- **Trocar MSC da AC:** É possível, através da troca de MSC, definir a localização da chave de uma AC. É importante ressaltar que a chave da AC não é trocada, e sim a localização de onde se encontra essa chave. Esta função é importante, por exemplo, em uma situação onde um MSC cadastrado apresente problemas de mal-funcionamento, sendo possível a troca por um MSC de backup que contenha a chave, podendo essa ter um identificador diferente.

3.1.8 Controle de Acesso

O SGC fornece três níveis principais de autenticação para o controle de acesso de suas funções, os perfis de acesso. Esses perfis, permitem que as funcionalidades do sistema sejam atribuídas para grupos de relevância diferenciados. O SGC divide estes perfis em Administração, Operação e Auditoria.

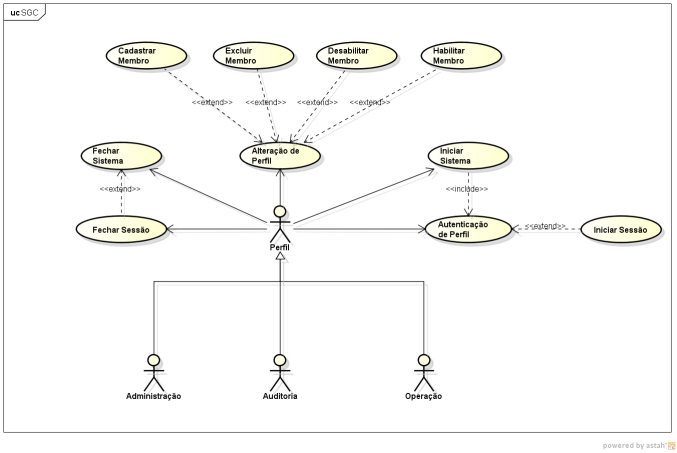


Figura 2 – “Casos de uso do SGC para os perfis de controle de acesso.”

- **Administrador:** Responsável pelo gerenciamento de AC e MSC, tais como criação, configuração e remoção.

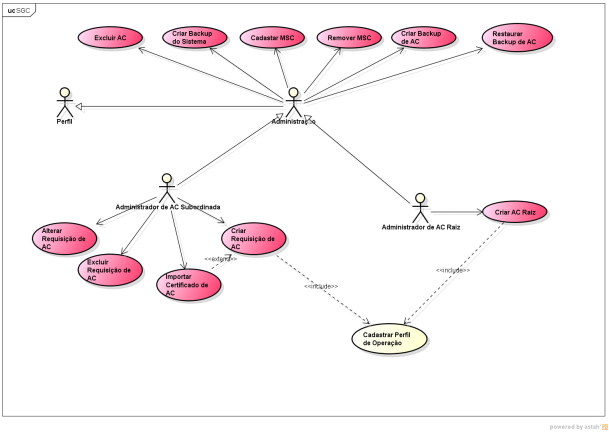


Figura 3 – “Casos de uso do SGC para perfil Administrador.”

- **Operador:** Responsável pelo gerenciamento de funções da AC, como emissão de certificados.

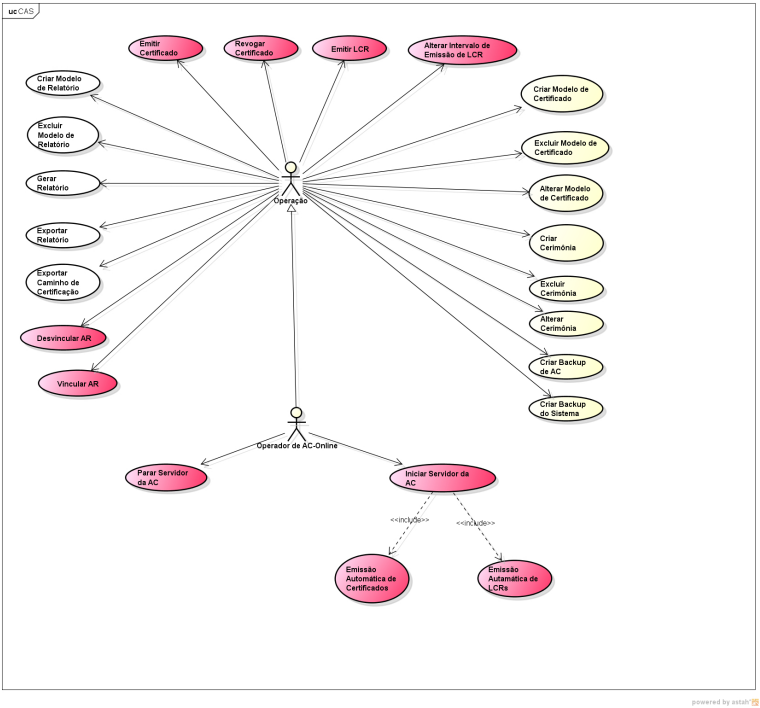


Figura 4 – “Casos de uso do SGC para perfil Operador.”

- **Auditor:** Responsável por fazer buscas (com ou sem filtros) em logs e relatórios e exportá-los.

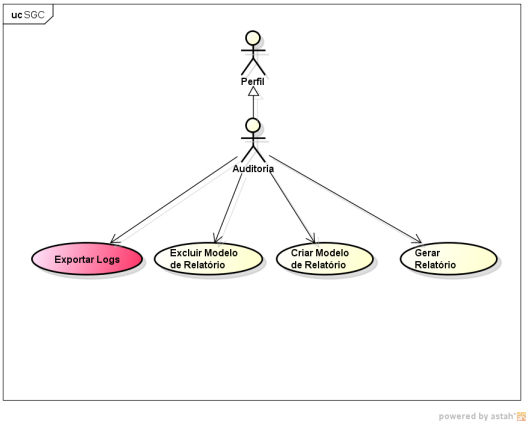


Figura 5 – “Casos de uso do SGC para perfil Auditor.”

Existe ainda um outro nível de acesso, o Instalador, mas este é resposável apenas pela manutenção técnica, instalação e configuração do sistema.

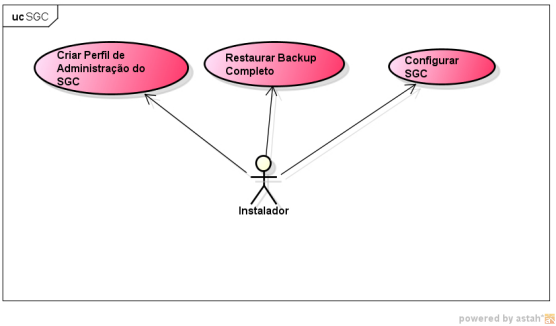


Figura 6 – “Casos de uso do SGC para perfil Instalador.”

3.1.9 Gerenciar Vínculos com Autoridades Registradoras

Os vínculos entre AC e AR são de extrema importância ao funcionamento de uma ICP. A AC deve possui uma lista de todas AR em que confia,

e vice-versa. Essa confiança entre autoridades se faz através do uso de certificados e identificadores.

- **Vincular Autoridade:** Importa certificado de AR. Contém uma lista de AR's confiáveis, com os respectivos certificados e identificadores;

3.2 SISTEMA DE AUTORIDADE REGISTRADORA

Um sistema gerenciador de AR's (SAR) funciona de forma semelhante aos *softwares* ramificados do SGC. Este sistema fornece funções básicas de um sistema gerenciador, comum à todas ramificações, assim como funções específicas à Autoridade de Registro.

Conforme o glossário ICP-Brasil (ICP-BRASIL, 2007a), uma AR é definida como uma entidade responsável pela interface entre o usuário e a AC. Sendo vinculada a uma AC, a AR tem por objetivo o recebimento, validação, encaminhamento de solicitações de emissão ou revogação de certificados digitais às AC e identificação, de forma presencial, de seus solicitantes. É responsabilidade da AR manter registros de suas operações. Pode estar fisicamente localizada em um mesmo ambiente que uma AC ou em ambientes segregados.

Um SAR deve prover funcionalidades que permitam a a criação e manutenção de uma AR, assim como fornecer recursos para que esta possa desempenhar seus objetivos. Para alcançar tais objetivos, o ITI define no Manual de Condutas Técnicas os requisitos necessários e ensaios para validação destes requisitos, que asseguram a correta implementação de um *software* de AR.

Detalhados a seguir estão alguns requisitos gerais, de segurança e auditoria retirados do MCT11 Vol.1 (ICP-BRASIL, 2010).

3.2.1 Criar Autoridade Registradora

Para a criação de uma Autoridade Registradora os passos são exatamente iguais com o de criação de uma Autoridade Certificadora Subordinada. A única diferença é que no certificado de uma Autoridade Registradora consta que ela não pode emitir outros certificados.

Os passos de criação de uma AR não serão detalhados, uma vez que está possui os mesmo passos que a criação de uma AC detalhada na seção anterior sobre Sistema gerenciador de certificados.

3.2.2 Gerenciar Agentes de Registro

O gerenciamento destes agentes acontece na Autoridade de Registro, eles devem possuir um cadastro com a AR tornando-os confiáveis para a mesma. A seguir será detalhado os passos para o gerenciamento de um Agente de Registro.

- **Cadastramento de Agente de Registro:**
  - **Importação de Certificado:** Para o cadastramento de uma Agente de Registro é necessário que o mesmo possua um certificado de pessoa fisica.
  - **Nome do Agente:** É necessário que o Agente de Registro possua um nome, este nome pode ser replicado o importante é que o mesmo só possua um certificado. A opção de um único certificado, garante que que não exista a replicação de agentes de registro.
- **Instalações Técnicas:** No cadastro do Agente de Registro, caso já exista alguma instalação técnica cadastrada na AR, pode-se efetuar a vinculação de um agente de registro. Não é obrigatório que o Agente de Registro esteja vinculado a uma Instalação Técnica.

- **Configurar Agente de Registro:** Após o cadastramento de um AGR, deve ser possivel modificar algum dos campos configurados anteriormente. O nome não pode ser mudado porém, caso seu certificado expirou o mesmo pode ser trocado por um novo.
- **Habilitar/Desabilitar Agente de Registro:** A exclusão de um AGR não é recomendada, pois caso seja necessário que a AR a liste agentes que não fazem mais parte da organização para se obter informações do mesmo. Portanto, é recomendável que possa habilitar e desabilitar agentes de registro.

3.2.3 Gerenciar Instalações Técnicas

A gerência de instalações técnicas serve para que a AR tenha um controle do local que os Agentes de Registros estão vinculados. Neste gerênciamento tem um ponto importante definido no manual da ICP-Brasil(ICP-BRASIL, 2007b), a qual especifica que o nome da Instalação Técnica deve ser único.

Para o gerênciamento de uma Instalação Técnica em uma AR deve conter os seguintes itens:

- **Cadastramento de Instalação Técnica:** Deve ser possível efetuar o cadastro de diversas Instalações Técnicas, respeitando a restrição de que o nome da Instalação Técnica é único.
- **Habilitar/Desabilitar Instalação Técnica:** Conforme especificado na subseção de Gerencia de Agentes de Registros, é recomendável que seja efetuado apenas as operações de habilitar e desabilitar invés de opções de excluir.

3.2.4 Gerenciar Vínculos com Autoridades Certificadoras

Outra tarefa que é composta na AR é o gerencimamento de vínculo confiável com Autoridades Certificadoras. Este vínculo serve para que AR

possa solicitar a emissão de certificados, revogação de certificados e solicitação de LCRs. O vínculo é efetuado em uma AR da seguinte forma: A AR importa o certificado de AC e a coloca em uma lista de AC's confiáveis, com os respectivos certificados, identificadores e endereço de repositórios que contenham informações relacionadas a AC vinculada. Esse repositório, e.g: Repositório *Lightweight Directory Access Protocol (LDAP)*, pode conter LCR's, lista de certificados emitidos, DPC's, etc.

Os itens a seguir descrevem de maneira detalhada o funcionamento das funções que esse gerênciamento deve fornecer:

- **Cadastramento de Autoridade Certificadora:** Deve ser possível efetuar o cadastro de diversas Autoridades Certificadoras, nesse cadastro deve ser informado o endereço *LDAP* da AC, assim como seu nome e o certificado de transporte para efetuar uma conexão segura com a AC.

- **Habilitar/Desabilitar Autoridade Certificadora:** Conforme especificado na subseção de Gerencia de Agentes de Registros, é recomendável que seja efetuado apenas as operações de habilitar e desabilitar invés de opções de excluir.

- **Emissão de certificado:** Para emitir um certificado a AR necessita estar vínculada a uma AC confiável. Após o Agente de Registro aprovar a solicitação de um certificado, a AR utiliza-se de uma forma segura de comunicação com a AC através do certificado de transporte e solicita a emissão do certificado.

- **Revogação de certificado:** Conforme mencionado no item anterior, a AR necessita uma conexão segura com a AC. Efetuado esta conexão segura, a AR solicita a revogação de um certificado para a AC.

- **Emissão de LCR:** A AR não emite uma LCR, ela efetua uma solicitação de LCR para as suas ACs vínculadas. Esta comunicação com as ACs deve ser efetuada de forma segura através do certificado de transporte.

3.2.5 Cadastrar Pedidos de Certificado

Outra tarefa de uma AR é o cadastro de pedidos de certificado, estes pedidos deve ser solicitado para que depois o Agente de Registro verifique a veracidade dos dados desta requisição.

O cadastro de pedidos deve ser efetuado através de um módulo público que é disponibilizado pela AR. Este módulo público pode ser Online ou OffLine.

3.2.6 Aprovar Pedido

A aprovação de um pedido deve ser efetuado pelo Agente de registro, após o mesmo verificar a veracidade dos dados que o usuário submeteu. Esta aprovação de pedido deve ser salvo na AR, para que no futuro possa ser efetuado uma consulta para saber quais já foram aprovados e emitidos.

3.2.7 Rejeitar Pedido

Conforme os dados forem apresentados no pedido de certificado, o Agente de Registro efetua a negação do pedido. Está rejeição também deve ser salva para que a AR tenha um controle sobre os pedidos que foram aprovados ou rejeitados. Também é necessário que seja salvo está ação para que caso aconteça uma auditoria, a AR demonstre todos os pedidos que foram aprovados ou negados.

3.2.8 Solicitar Emissão de LCR

Conforme mencionado na subseção de Gerenciar Vínculo com Autoridades Certificadoras, a solicitação de LCR é feita através de um canal seguro com a Autoridade Certificadora que a AR possui vínculo.

Está LCR que é emitida deve estar assinada pela AC, garantindo a con-

fiabilidade. A AR não deve assinar a LCR apenas repassar para uma aplicação ou usuário final que a solicitou.

3.2.9 Controle de Acesso

Para garantir a integrinidade de acessos em uma AR, é necessário que a mesma possua modos de autenticação para que Operadores e Agentes de Registro cadastrados no sistema a utilizem.

Está autenticação pode ser feita de diversas formas, a mais usual para operadores é através de *SmartCards*, que devem ser configurados na criação de uma AR. Já a maneira mais usual para Agentes de Registro é a verificação de seus certificados digitais, uma vez que os mesmos podem não estar no mesmo espaço físico que a AR.

3.2.9.1 Agentes de Registro

Cada agente de registro está vinculado a uma Instalação Técnica e possui um certificado cadastrado. Este certificado cadastrado serve para que o agente seja autenticado no sistema, e suas ações como agente, registrados no histórico.

3.2.9.2 Operadores da AR

Os operadores da AR possuem um *SmartCard* para autenticação no sistema. Estes operadores encontran-se no mesmo espaço físico que a AR e suas responsabilidades incluem gerenciar Agentes de Registros, Gerar Logs e gerenciar Instalações Técnicas.

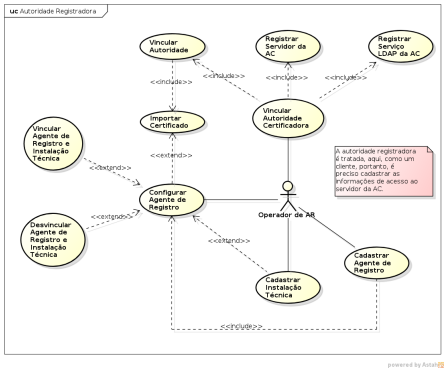


Figura 7 – “Casos de uso do SAR para Operadores de AR.”

3.3 MÓDULO PÚBLICO

A aplicação de módulo público é essencial para a padronização na área de requisições de certificados. Como explicado na fundamentação teórica deste trabalho, a criação de uma requisição de certificado necessita de uma interface mais amigável.

Atualmente através do *OpenSSL* é possivel gerar requisições por linha de comando, conforme a figura 8. Contudo o usuário pode ficar confuso no preenchimento de determinados campos e que se errôneamente preenchidos podem acarretar na possível revogação do certificado emitido.

3.3.1 Solicitar Certificado

Uma das funções do módulo público oferecido pela AR, é a solicitação de certificado. Nesta solicitação de certificado é requerido do usuário informações básicas para a indentificação do mesmo.

Após o usuário solicitar o certificado o mesmo deve comparecer a AR para a verificação dos dados. Por isto é importante os dados para indentificação do usuário, para que após uma análise inicial do pedido a AR indentifique o

```
gustavo@gustavo-Vostro1510:~$ openssl genrsa -des3 1024 > test.key
Generating RSA private key, 1024 bit long modulus
............++++++
.....................................++++++
e is 65537 (0x10001)
Enter pass phrase:
Verifying - Enter pass phrase:
gustavo@gustavo-Vostro1510:~$ openssl req -new -key test.key -out requi.req
Enter pass phrase for test.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:BR
State or Province Name (full name) [Some-State]:Santa Catarina
Locality Name (eg, city) []:Florianopolis
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Teste
Organizational Unit Name (eg, section) []:LabTest
Common Name (eg, YOUR name) []:Teste
Email Address []:teste@teste.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
gustavo@gustavo-Vostro1510:~$
```

Figura 8 – “Exemplo de criação de uma requisição de certificado com OpenSSL.”

requerinte.

3.3.2 Gerar Requisição de Certificado

O Módulo Público também oferece a geração de Requisição de Certificado, este passo deve ser efetuado depois do passo exemplificado na subseção Solicitar Certificado.

Nesta funcionabilidade do módulo público deve ser inserido os dados para a criação de uma requisição, os dados que contém nesta requisição podem variar de acordo com o país que oferece a opção de emitir certificado. Tomando como exemplo a geração de requisição no Brasil, o usuário irá inserir CPF, RG, PIS/PASSEP entre outros dados.

3.3.3 Importar Certificado

A funcionabilidade de importar certificado oferecida no módulo público, serve que após os passos apresentados nas subseções anteriores sejam efetuados e que a AR tenha aprovado e emitido o certificado.

O usuário possa importar o seu certificado para um *keystore* ou um Agente de Registro o coloque em um *token*.

3.3.4 Solicitar Revogação de Certificado

Caso o usuário comprometa o seu certificado, é necessário que o mesmo solicite a revogação do certificado. Está opção deve ser oferecida no módulo público.

Na solicitação de revogação o usuário deve especificar o motivo pelo qual está solicitando a revogação.

3.4 ASSINADOR

O assinador no contexto de ICP, é uma aplicação criptográfica que faz uso da certificação digital para assinar e validar digitalmente documentos. A assinatura digital deve prover autenticação, integridade de dados e não-repudiação do assinante. Isto é obtido através de uma transformação criptográfica dos dados. Já a validação é feita no intuito de verificar a validade destas assinaturas. Esta aplicação deve estar em conformidade com as especificações estabelecidos pela ICP-Brasil, e deve ser capaz de manipular certificados digitais X.509 versão 3 definidos pelo ITU-T.

No momento da assinatura do documento eletrônico, o *software* assinador adiciona as informações contidas no certificado ao documento. Isto faz com que seja gerado uma identificação única que une o portador do certificado ao documento assinado digitalmente.

Segundo o MCT4 Vol.1 (ICP-BRASIL, 2007b), além de implementar

os requisitos gerais de certificação digital, um *software* de assinatura digital deve implementar os seguintes requisitos específicos:

3.4.1 Assinar

O assinador gera um resumo criptográfico (*hash*) da mensagem utilizando algoritmos. Esse resumo passa então pelo um processo de cifragem, com a assinatura da mensagem pela chave privada do autor (signatário) e a junção do *hash* cifrado à mensagem original.

3.4.2 Verificar Assinatura

O assinador gera um novo resumo criptográfico da mensagem. O resumo original é obtido através de decifragem da assinatura digital do documento, estes dois resumos são então comparados afim de analisar a igualdade. Caso haja a igualdade entre os resumos, pode-se confirmar a autoria e integridade da mensagem.

4 MODELAGEM

Neste capítulo serão listados todos artefatos de engenharia de software gerados durante o desenvolvimento deste trabalho. Esses artefatos se dividem em: requisitos, casos de uso, diagramas de atividade e componentes. Apesar de especificados neste documento como uma forma "final" do trabalho, esses artefatos estão sujeitos a alterações, conforme são utilizados e testados em aplicações reais. Dessa forma, o *framework* não se engessa às aplicações previstas inicialmente.

4.1 REQUISITOS

O levantamento de requisitos para a implementação de um *framework* é uma tarefa, no mínimo, trabalhosa. As aplicações, que o *framework* abrange, nivelam o número de requisitos e a dificuldade de especificá-los. Neste trabalho, cada requisito foi descrito em um alto nível de abstração, reduzindo sua complexidade e permitindo um melhor entendimento de como o *framework* deve ser organizado, para suprir as necessidades das aplicações de uma ICP. As especificações mais minuciosas são feitas no formato de casos de uso, na seção 4.2.

A seção 4.1.1 lista os requisitos gerais do *framework*, dando uma visão macroscópica, onde pode ser vistoo como ele será dividido em módulos e a forma que será implementado. As outras seções apresentam requisitos mais técnicos, especificando, principalmente, os modelos que devem existir no *framework*.

4.1.1 Gerais

4.1.1.1 Paradigma de programação

Todo código deve ser feito a partir do paradigma de programação de orientação à objetos e, de forma complementar, fazer uso dos conceitos de programação orientada à componentes. As classes devem ser desenvolvidas de forma que possam ser reaproveitadas diretamente ou através de reimplementação de funções. Deve ser estabelecido uma componentização ótima, ou próxima de ótima, afim de granularizar a complexidade do projeto em componentes específicos de menor complexidade, aumentar a especialização e o reuso do código.

4.1.1.2 Módulo de Interface Gráfica

O *framework* deve possuir um módulo de interface gráfica, que conterá janelas pré-desenhadas em QT. As janelas fornecidas como parte integrante do *framework* serão adaptáveis a diversos contexto de aplicação, com componentes gráficos flexíveis de diferentes granularidades que concedem ao *framework* uma customização gráfica.

4.1.1.3 Módulo de Persistência

O *framework* deve possuir um módulo de persistência, que conterá funções utilizáveis em diversas aplicações no contexto de ICP. Esse módulo não deve ter nenhuma dependência com a forma que os dados são persistidos, e sim com o que e quando algo deve ser persistido.

4.1.1.4 Módulo de Domínio

O módulo de domínio contém as abstrações das entidades existentes em uma ICP, como, por exemplo: Autoridades Certificadoras, Autoridades Registradoras, Certificados, Certificados Revogados e Listas de Certificados Revogados.

4.1.2 Autoridade Certificadora

O *framework* deve possuir uma abstração para uma Autoridade Certificadora genérica, que possui as três funções básicas de uma AC: Emitir Certificado, Revogar Certificado e Emitir Lista de Certificados Revogados. As funções de emissão devem ser adaptáveis a diversos padrões de ICP (ex: X.509), tipos de chaves (ex: RSA, ECDSA, DSA), algoritmos de resumo criptográfico (ex: SHA1, SHA2) e interfaces com módulos de segurança (ex: engine openssl, PKCS#11, chave em arquivo). A AC também deve usar uma interface de comunicação com a persistência, permitindo que o desenvolvedor defina a forma de persistir de dados (base de dados, arquivos em disco, memória ram).

4.1.3 Autoridade Registradora

O *framework* deve possuir uma abstração para uma Autoridade Registradora genérica, que possui todas suas funções básicas, como: Aprovar Pedido de Certificado, Rejeitar Pedido de Certificado, Solicitar Emissão de Certificado, Solicitar Revogação de Certificado e Solitar Emissão de LCR.

4.1.4 Formato X.509

O *framework* deve dar suporte ao padrão X.509 e deve possuir abstrações para todas estruturas definidas por ele.

4.1.5 Templates de Certificado

O *framework* deve disponiblizar classes para criação de templates de certificados. Estes templates devem ser utilizados para o pré-preenchimento de construtores (*builders*) de certificado.

4.1.6 Métodos de Acesso à Chave

O *framework* deve disponibilizar, pelo menos, três formas de acesso à chaves privadas: engine openssl, interface PKCS#11 e acesso em disco (sem uso de módulo criptográfico). Porém, o *framework* deve, também, ser adaptável a outros padrões.

4.1.7 Smartcards

O *framework* deve disponibilizar classes de abstração para *smartcards*.

4.1.8 Extensões de Certificado da ICP-Brasil

O *framework* deve disponiblizar todas extenões de certificados disponíveis dentro da ICP Brasil. Porém, deve se manter adptável para qualquer outro tipo de extensão que possa existir.

4.1.9 Sistema de Logs

O *framework* deve disponibilizar ferramentas para geração de logs seguros, logs que são assinados.

4.1.10 Suporte a Algoritimos Criptográficos

O *framework* deve dar suporte a diversos algoritmos criptográficos, principalmente aos mais usados atualmente. Além disso, o *framework* deve estar preparado para o surgimento de novos algoritmos, para que possa integrá-los ao código sem grande impacto em sua estrutura.

4.2 CASOS DE USO

O *framework* dita o fluxo de dados, estabelece um padrão de como a aplicação deve proceder na implementação dos seus casos de uso específicos. Dessa forma, os casos de uso mais importantes de um *framework* são aqueles que aparecem em mais de uma aplicação. Para fazer uma analise de quais são esses casos de uso, os autores levaram em consideração as aplicações de Autoridade Certificadora Raiz, Subordinada e Final, Autoridade Registradora e Módulo Público. A seguir, serão especificados os casos de uso de cada uma dessas aplicações. No final da seção, é feita uma análise sobre quais casos de uso devem receber mais atenção na implementação do *framework*.

Os casos de uso de um *framework* são definidos pela aplicação em desenvolvimento. Este define apenas os serviços fixos que serão utilizados na especificação de casos de uso das aplicações.

A implementação destes casos de uso em conjunto com as abstrações fornecidas pelo *framework* irá gerar uma aplicação final, um SGC.

4.2.1 Administração

Os casos de uso referente a uma Autoridade de Registro e de um AC-Final são os mesmos que de uma AC-Subordinada.

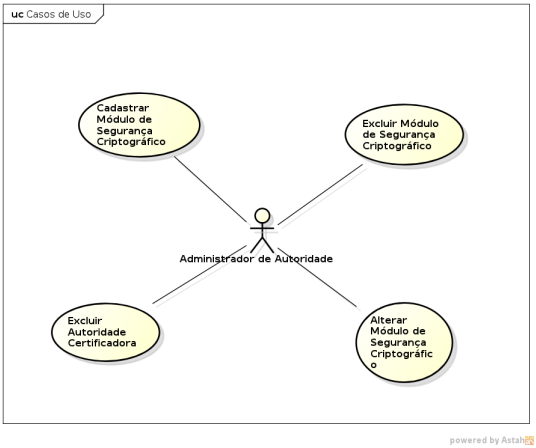


Figura 9 – “Casos de uso do *framework* para Administração.”

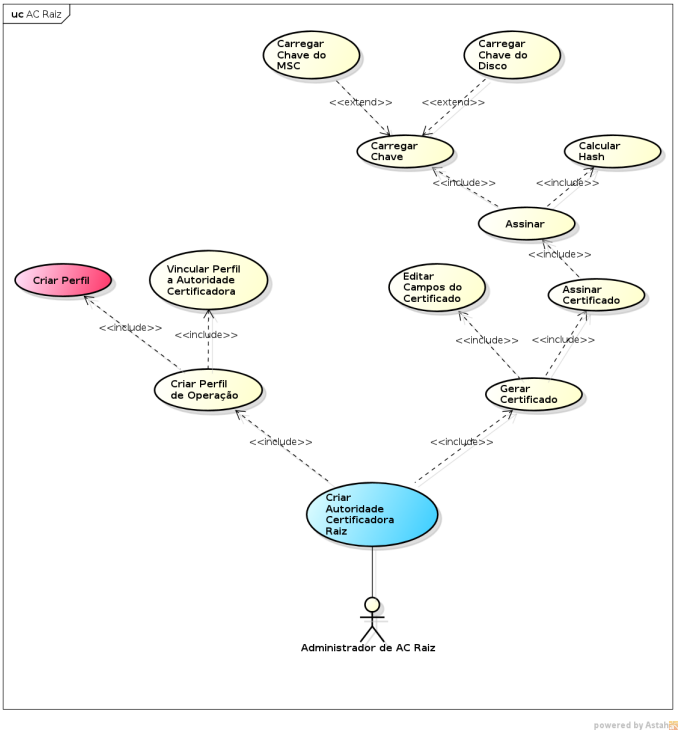


Figura 10 – “Casos de uso do *framework* para Administração de uma AC-Raiz.”

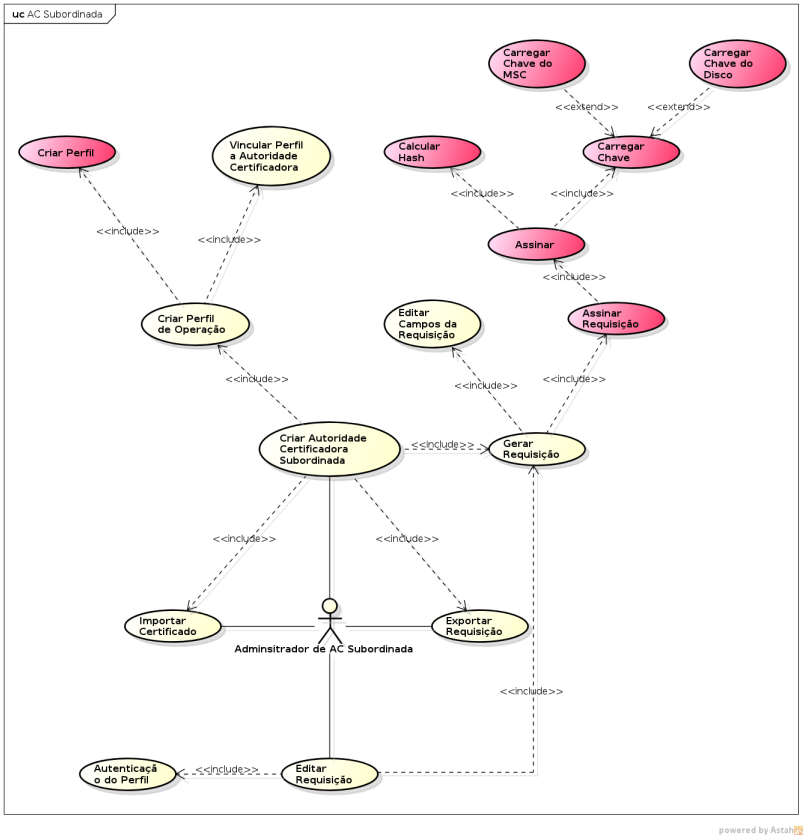


Figura 11 – “Casos de uso do *framework* para Administração de uma AC-Subordinada.”

4.2.2 Auditoria

Os casos de uso de auditoria do sistema estão representados na figura.

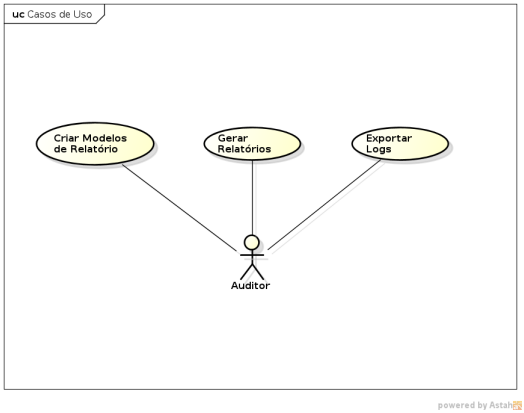


Figura 12 – “Casos de uso do *framework* para Auditoria.”

4.2.3 Operação

Dentre os casos de uso de operação, temos os casos de uso de AC Raiz, AC Subordinada, AC Final e AR. Estão representados nas figuras 13, 14, 15 e 16.

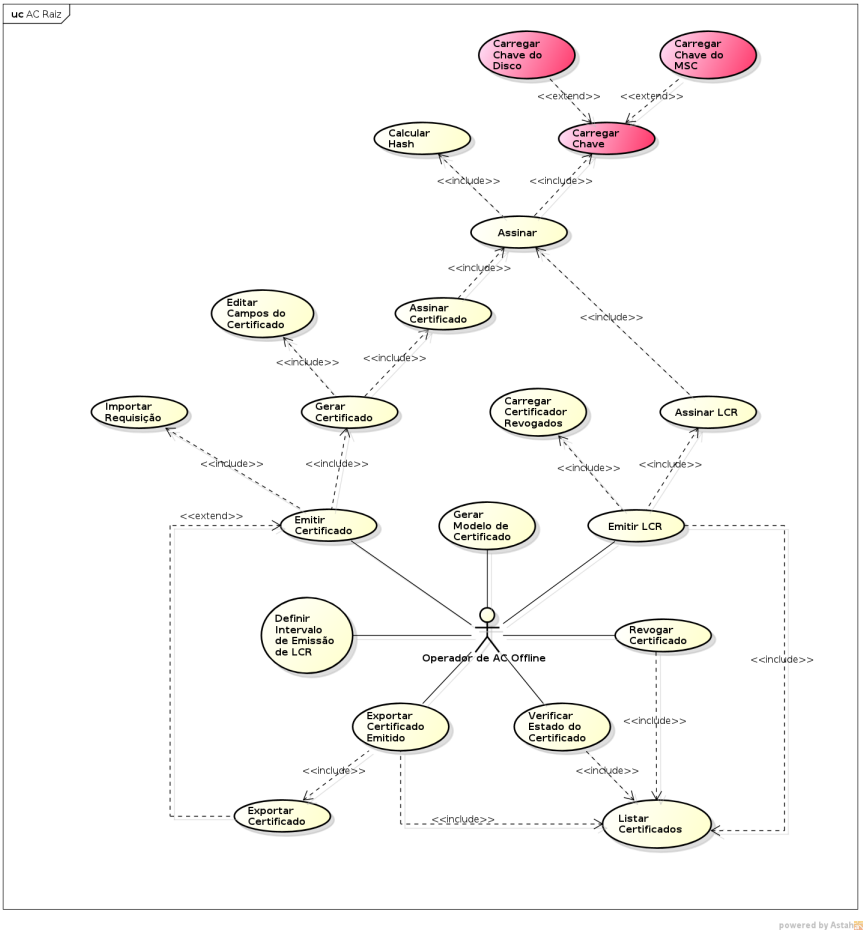


Figura 13 – “Casos de uso do *framework* para Operação de uma AC Raiz.”

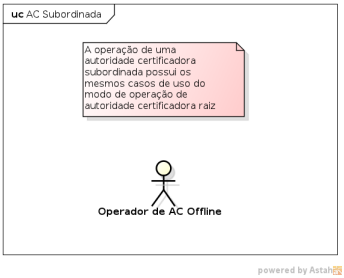


Figura 14 – “Casos de uso do *framework* para Operação de uma AC Subordinada.”

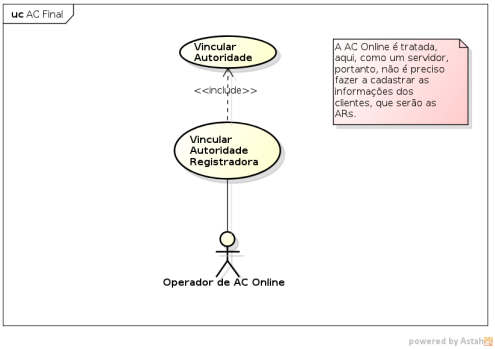


Figura 15 – “Casos de uso do *framework* para Operação de uma AC Final.”

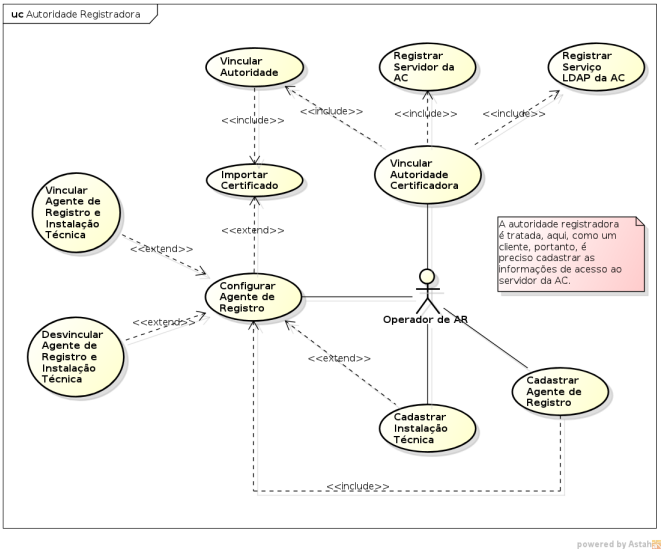


Figura 16 – “Casos de uso do *framework* para Operação de uma AR.”

4.3 DIAGRAMAS DE ATIVIDADES

As funções comuns ao sistema tornam-se visíveis a partir da análise dos casos de uso. Para melhor expressar estes casos de uso, foram criados os diagramas de atividade que representam os fluxos desses processamentos.

Os diagramas contendo os fluxos de criação de AC Raiz, Subordinada e Final podem ser encontrados na seção de anexos deste trabalho, assim como outras operações relacionadas a AC.

4.4 COMPONENTES

Conforme explicado na seção de fundamentação teórica, componentes são partes distintas de um *software* desenvolvidas de forma independente (módulos) que realizam conexões com outros componentes do próprio *software*, utilizando-se de serviços e recursos definidos por interfaces de ligação

bem definidas.

Na implementação da LibPKI foi utilizado o conhecimento sobre componentes para efetuar uma modelagem mais consistente. Desta modelagem surgiram diversos componentes que os autores deste trabalho analisaram como a melhor solução. Os componentes são os seguintes:

4.4.1 ASN1

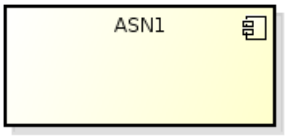


Figura 17 – "Figura referente ao componente ASN1."

De acordo com o que foi explicado na fundamentação deste trabalho, a notação *ASN.1* define a estrutura dos dados. Portanto, a idealização deste componente se deu com o intuito de especificar classes que sejam especializadas na estrutura dos dados definida.

Este componente, por exemplo, possui uma classe chamada Asn1BitString. Vemos dessa maneira a especialização de uma classe para tratamento do dado. Esta classe, assim como outras deste componente, servem para tratamento de dados e não para criação. A criação deste dado é feita no componente *Public Key Infraestrutucture*.

4.4.2 Authetication

O componente *Authetication* tem como principal atividade oferecer a autenticação no sistema. A autenticação é responsável por iniciar as atividades do *software*. Este componente irá prover a autenticação do sistema que poderá ser feito através da utilização de um *smartcard*.

Para prover as funções de autenticação, o componente faz uso dos serviços disponibilizados através da interligação com o componente *Secret*, que será descrito na seção seguinte. A função do componente *Secret* para com este é a identificação dos usuários capazes de se autenticar no sistema. Conforme mencionado, o componente irá prover a capacidade de autenticação do sistema, para isso aconteça pode ser necessário que um ou mais usuários do sistema se autentiquem, e neste tipo de autenticação é utilizado a noção de segredo compartilhado.

4.4.3 Secret

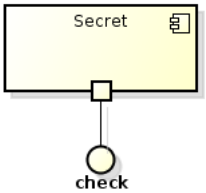


Figura 18 – "Figura referente ao componente Secret."

O componente *Secret* é o responsável por criar o segredo compartilhado, necessário para que os usuários se autentiquem no sistema. Este segredo compartilhado é gerado pelo sistema de acordo com o número de usuários no sistema. Caso exista apenas um usuário se autenticando, não existe a necessidade de criar um segredo compartilhado.

Este componente fornece uma interface que contém métodos, que efetuam a chamada de classes internas responsáveis pela geração e pla montagem de segredo compartilhado. Por está caracteristica única e necessária para a segurança de um SGC, é que esta parte do *software* pode ser encapsulado em um componente.

Quando existe apenas um usuário no sistema, não existe a possibi-

lidade de gerar o segredo compartilhado. Porque para efetuar a divisão do segredo é necessário que existe mais de um usuário no sistema.

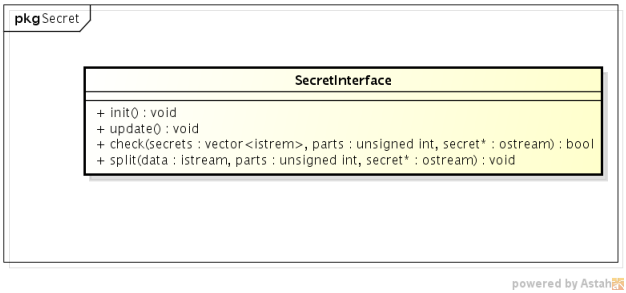


Figura 19 – “Figura referente as interfaces providas do componente *Secret*.”

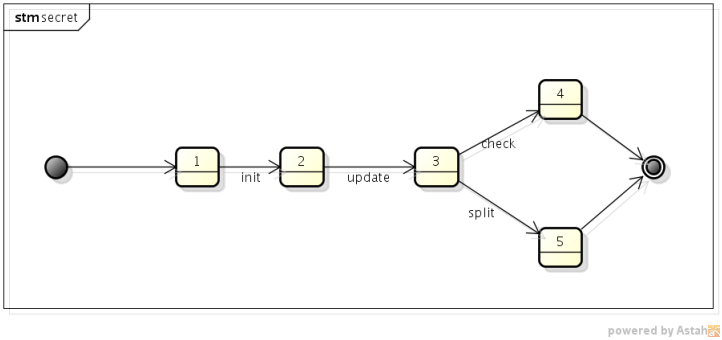


Figura 20 – “Figura referente a máquina de estados do componente *Secret*.”

4.4.4 Cryptography

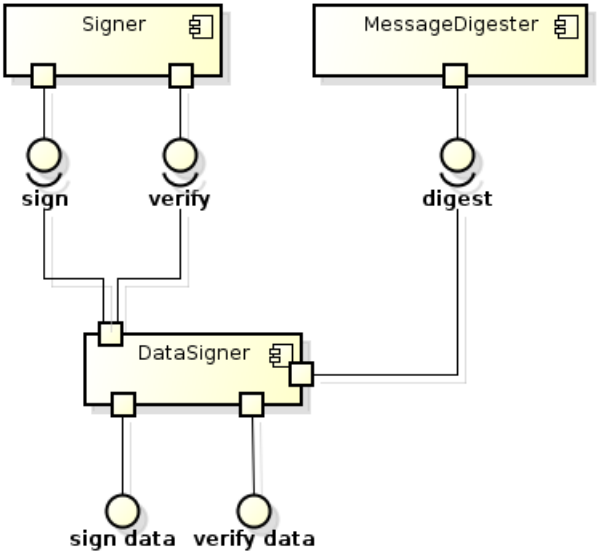


Figura 21 – “Figura referente ao componente Cryptography.”

O componente *Cryptography* é responsável pelas funções criptográficas do *framework*. Ele contempla todas as funções criptográficas necessárias para utilizações em ICPs, além de possibilitar a extensão destas funções para adequação ao *software* que irá ser implementado.

Encontram-se dentro deste componente, funções específicas para cifragem, algoritmos implementados e estruturas que viabilizam a utilização destes.

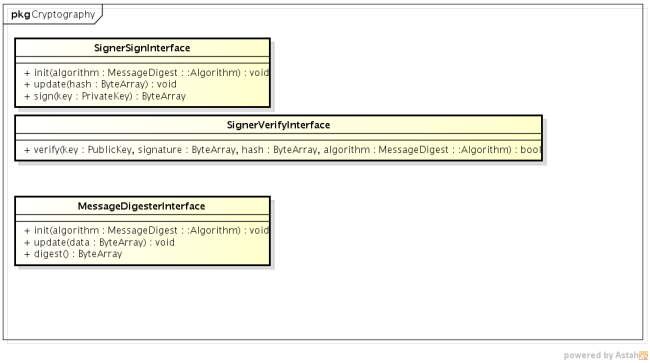


Figura 22 – “Figura referente as interfaces providas do componente Cryptography.”

Pode-se perceber na figura 22 que está fornece as interfaces necessárias para que o componente possa ser utilizado. Temos no componente *Signer*, duas interfaces a assinatura e a verificação de um dado assinado.

Para que seja utilizado a assinatura, é necessário primeiramente efetuar a inicialização da interface chamando o método *init* com os parametros necessários, após isso é necessário chamar o *update* para que o mesmo receba o dado que irá ser assinado e finalmente o método de assinatua a qual recebe chave privada, conforme as figuras 24 e 25.

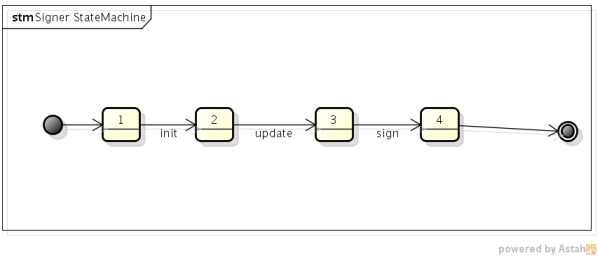


Figura 23 – “Figura referente a maquina de estados do componente *Cryptography*.”

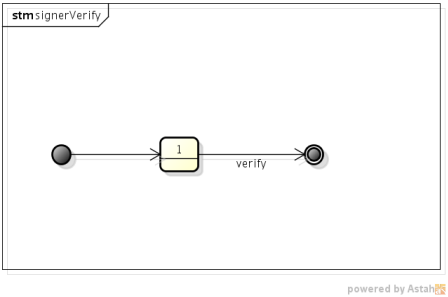


Figura 24 – “Figura referente a maquina de estados do componente *Cryptography*.”

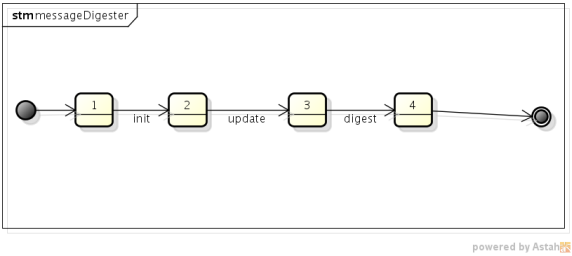


Figura 25 – “Figura referente a maquina de estados do componente *Cryptography*.”

4.4.5 Exception

Componente para o tratamento de exceções para a questão de persistência, modelo e interface. Além de prover exceções para que o usuário do *framework* possa utilizar, estas exceções também são chamadas dentro dos outros componentes descritos na seção de Modelagem.

Pode-se destacar que as classes de exceções estão respectivamente dentro de seus componentes: model, persistence e interface. Não sendo um componente a parte, pois cada exceção trata erros referente ao seu componente.

4.4.6 Public Key Infrastructure

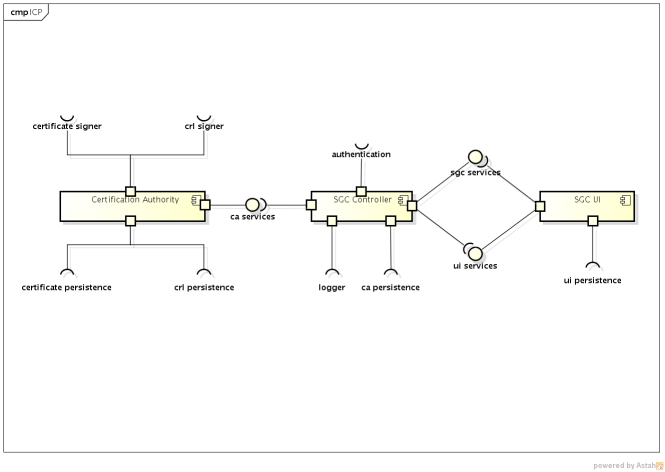


Figura 26 – “Figura referente ao componente à estrutura de uma AC Raiz.”

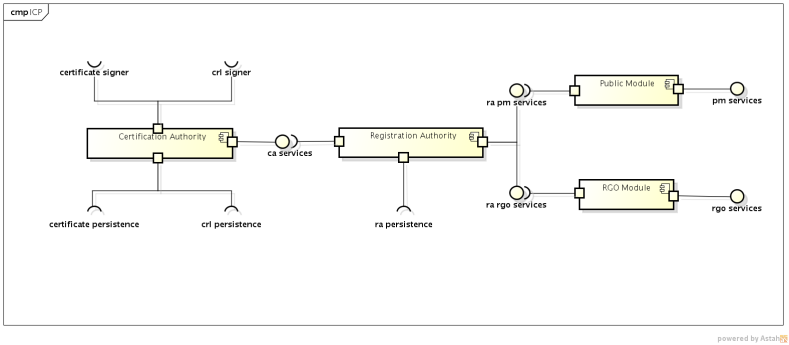


Figura 27 – “Figura referente ao componente à estrutura de uma AC final.”

Um dos principais componentes para o *framework*. Responsável por toda a parte de desenvolvimento para a ICP. Contém classes que são especializadas no tratamento de SGC, SAR, classes para tratamento de certificados e outras funções contempladas no âmbito de ICP.

Public Key Infraestructure é um dos componentes mais importantes no *framework*, o seu próprio nome explica o grau de sua importância. É lógico que somente ela não constitui todo o *framework* por questões de ela necessitar as funções de outros componentes já descritos nessa seção como: algoritmos de criptográficos, autenticação no sistema, tratamento de exceções e outras funções.

4.5 DIAGRAMA DE BASE DE DADOS

A parte de persistência contém diagramas específicos, como o diagrama de modelagem de base de dados. Este diagrama é essêncial para o entendimento do funcionamento do componente de persistência.

Este diagrama de base de dados é baseado nos requisitos levantados, na compreensão dos autores sobre engenharia de software e sobre o conhecimento específico de aplicações em ICP. Considerando o modelo o mais simples para o funcionamento de um SGC, pois contém apenas os aspectos básicos do funcionamento de uma AC, que são: Emitir Certificado, Revogar Certificadoe, gerar Lista de Certificados Revogados e salvamento de dados sobre a AC, a figura 80, que se encontra nos anexos, representa este modelo.

Não existe somente a opção de salvar os dados em um SGBD é possivel também salvar os dados em arquivo ou na memória, claramento por motivos de segurança este dois casos não são recomentados.

5 PROCESSO DE DESENVOLVIMENTO DA LIBPKI

Nesse capítulo os autores propõem a organização do desenvolvimento para o framework LibPKI, isso inclui definição das etapas de desenvolvimento, organização da equipe, segmentação de desenvolvimento, acompanhamento da implementação, gerenciamento de qualidade de código, configuração do ambiente de trabalho, aplicação de técnicas de desenvolvimento e produção de documentação.

5.1 ORGANIZAÇÃO DA EQUIPE

A proposta de organização da equipe segue um modelo já utilizado em outros projetos do LabSEC, no qual são definidos três tipos de gerente: gerente de projeto, gerente de qualidade e gerente de configuração, com suas funções e responsabilidades detalhados a seguir:

- **Gerente de Projeto:** O gerente de projeto é responsável pela organização geral do projeto. É o único cargo que não precisa fazer parte da implementação efetiva da solução, porém é responsável pelas tomadas de decisões e pela tramitação com clientes e interessados. Entre suas tarefas encontram-se:
  - Definição dos membros que serão gerentes de qualidade e configuração;
  - Organização da soluções em um nível elevado de abstração;
  - Levantamento de requisitos e organizção da modelagem do sistema;
  - Criação de tarefas e definição de responsáveis;
  - Produção da documentação de desenvolvimento;
  - Atendimento de Clientes;

- **Gerente de Qualidade:** O gerente de qualidades é responsável pela qualidade do código produzido no desenvolvimento da solução. Suas tarefas e responsabilidades são:
  - Revisão dos códigos entregues pelos desenvolvedores;
  - Avaliação da lógica de programação;
  - Avaliação de concordância entre a modelagem e a implementação;
  - Avaliação da usabilidade da solução;
  - Definição de tarefas de melhorias de código;
  - Finalizar tarefas, quando entregues;
- **Gerente de Configuração:** O gerente de configuração mantém todo conhecimento tecnológico da solução. Suas tarefas e responsabilidades são:
  - Estudar e sugerir tecnologias úteis ao desenvolvimento da solução;
  - Conhecer todas tecnologias que serão usadas no desenvolvimento;
  - Configurar novos ambientes de desenvolvimento;
  - Solucionar problemas relativos a infraestrutura;

5.2 TÉCNICAS DE DESENVOLVIMENTO

5.2.1 *Scrum*

Scrum é um processo ágil de desenvolvimento de *software*. O progresso de um projeto desenvolvido com scrum se dá através de uma série de iterações chamadas *Sprints*. Cada *sprint* tem duração aproximada de 2 à 4 semanas. Antes de ser considerado um processo completo ou uma metodologia, o *scrum* é um *framework*. Então, ao invés de fornecer uma solução completa, muito é deixado ao grupo de desenvolvimento de *software*, pois este sabe como melhor resolver os problemas a eles apresentados.

O grupo de desenvolvimento realiza reuniões diárias, conhecidas como "reuniões de pé", onde é discutido o andamento do projeto e são elaborados planos de ação. Este método ágil depende de um grupo de desenvolvimento auto-organizado e multidisciplinar. Não há especificamente um indivíduo que lidera e designa tarefas, e sim uma decisão coletiva acerca dos problemas. Deste grupo dois indivíduos se destacam, o *ScrumMaster* e o *Product Owner*, o primeiro auxilia o grupo no desenvolvimento e o segundo garante a correta definição do produto.

O termo "Scrum" (abreviação de "Scrummage") originou-se de uma formação de Rugby, onde os jogadores se posicionam juntos e de pé, e há uma disputa pela bola e retomada de jogo após uma penalidade.

Scrum não é um método exclusivo ao desenvolvimento de *software*. É um método para gerenciar o desenvolvimento do produto que pode ser envolto por qualquer tecnologia específica, incluindo *software*. O Scrum como existe hoje amadureceu desde o seu início na década de 1980 no Japão. (TAKEUCHI; NONAKA, 1986)

5.2.2 Desenvolvimento Orientado à Testes

O desenvolvimento orientado à testes (do inglês *Test-Driven Development* – TDD) é uma técnica de desenvolvimento orientada a testes, que tem como um de seus objetivos antecipar a identificação e correção de falhas durante o desenvolvimento. O principal objetivo do TDD é possuir um código limpo que funcione. Os testes unitários são amplamente utilizados e realizados no código fonte da aplicação, aprimorados por refatoração até que os objetivos da funcionalidade fiquem de acordo com as especificações desejadas. O TDD faz parte de um grande paradigma de projeto de *software*, conhecido como *Extreme Programming* (XP), e com o uso adaptado ao contexto deste trabalho.

Aplicações desenvolvidas com esta técnica possuem qualidade superior a desenvolvidas com outros métodos em um mesmo tempo. A natureza

do TDD assegura que todas as unidades da aplicação foram testadas visando o seu melhor funcionamento. A qualidade do *software* está ligada diretamente a qualidade dos testes.

O ciclo de desenvolvimento orientado a testes se resume a: Adicionar um teste – Executar todos os testes ver se o novo passa – Escrever código – Executar os testes automatizados e ver seu sucesso – Refatorar código – Repetir. (BECK, 2003)

5.2.3 CPPUnit

O *CPPUnit* é um *framework* de testes unitários para a linguagem C++, semelhante ao *JUnit* para java. Juntamente com a técnica de TDD o *CPPUnit* é aplicado na criação de testes unitários (FEATHERS; LEPILLEUR, 2002).

A aplicação de testes unitários tem sido considerado de suma importância para grandes projetos de *software* devido a sua fácil aplicação e resultados de fácil interpretação. Neste contexto, o *CPPUnit* traz uma segurança para desenvolvedores na linguagem C++, sua facilidade em criar testes, agilidade na execução e sua estabilidade faz com que cada vez mais este framework seja utilizado (FEATHERS; LEPILLEUR, 2002).

5.2.4 *Clean Code*

Clean code ou “código limpo”(da tradução direta), é um conjunto de técnicas e métodos para o desenvolvimento de código reusável e bem estruturado. Existem inúmeras definições de *clean code*, tanto quanto o número de programadores, pois cada um tem uma definição do que é um código limpo. Entretanto, estas técnicas buscam enfatizar as qualidades comum a todas as definições, tais como: simplicidade, elegância, facilidade de leitura, etc. São muitas as propriedades de *clean code*, algumas se aplicam a todas as linguagens de programação e domínios de problemas, outras dependem apenas de linguagens de programação específicas.

> *"Código limpo não é somente desejável – é necessário. Se o código não estiver limpo, o desenvolvimento decai com o tempo."* (MARTIN, 2009)

O processo de refinamento de *software* consiste em uma constante análise do código fonte, em busca de determinados padrões e com o objetivo da refatoração de código. Estes padrões analisados pelo *clean code* vão desde a nomenclatura de variáveis e funções até a formatação de código e uso correto de comentários.

O objetivo principal do *clean code* é a qualidade do código. O uso de boas práticas de programação e adequação as regras garantem que o código vá realmente expressar a finalidade do *software*. O resultado obtido com a aplicação destas regras implica diretamente em uma maior produtividade da equipe, menos tempo procurando e corrigindo possíveis *bugs*, no bem-estar do programador, pois alterações em código limpo são simples e objetivas.

> *"A arte de programar é, e sempre foi, a arte de desenhar linguagens. Grandes programadores quando pensam em sistemas, pensam em histórias para serem contadas em vez de programas para serem escritos. Eles usam as ferramentas que a linguagem de programação escolhida lhes proporciona para construir uma linguagem muito mais rica e expressiva que possa ser usada para contar a história."* (MARTIN, 2009)

5.2.5 Integração Contínua

O termo "Integração contínua"originou-se do processo de desenvolvimento *Extreme Programming – XP*, como uma de suas doze práticas. A integração contínua é uma prática de desenvolvimento de software, que ficou em evidência devido ao impacto do uso das metodologias ágeis, normalmente pelo uso de XP, e no contexto deste trabalho, pelo *Scrum*. A técnica consiste em integrar frequentemente o trabalho de uma equipe de desenvolvimento. Esta prática acarreta em menores ocorrências de problemas relacionados a integração de código.

A cada integração, um processo automatizado efetua o *build* através de tarefas automáticas e gera instantaneamente um *feedback*. Para que isso seja possível, um servidor de integração deve ser o responsável por manter as tarefas automatizadas e estar configurado adequadamente. Neste servidor é possível fazer a instalação de *plugins* que atendem a diferentes funcionalidades, desde análise estática de código à geração de documentação de código. Ao final de cada integração junto ao servidor, obtemos uma versão funcional da aplicação pronta a entrar em produção.

Existem duas forma de integração contínua:

- Síncrona: Novas integrações são impedidas enquanto a integração corrente não estiver finalizada. Neste modo há a necessidade que os desenvolvedores trabalhem juntos, e sejam informados do término de uma integração. Este modelo força a os desenvolvedores a verificar imediatamente falhas na integração, porém há uma segurança maior no conteúdo adicionado ao repositório de código, localizado localmente em um desktop por exemplo.

- Assíncrona: Este modelo permite aos desenvolvedores estarem geograficamente distribuídos. O uso desta forma de integração é recomendado, por exemplo, em aplicações *open source*, onde há uma participação ativa da comunidade. A gestão é feita através de um servidor de integração, que coordena a requisição concorrente de integrações.

A eficiência da integração síncrona é superior devido ao fato do *feedback* ser imediato, alertando rapidamente o desenvolvedor sobre problemas na integração e mantendo o foco para esta atividade.

5.3 FERRAMENTAS E TECNOLOGIAS

5.3.1 Red Hat Enterprise Linux

É proposto o uso do Red Hat Enterprise Linux, RHEL, pois possui um suporte eficiente e é um sistema operacional consolidado e robusto. Outro motivo para que RHEL seja utilizado é a compilação estável de bibliotecas que devem ser usadas, como a LibcryptoSec.

Atualmente é utilizado nos computadores da ICP-Brasil o RHEL, como o *framework* proposto neste trabalho tem como objetivo a sua utilização na ICP-Brasil é recomendado que seja utilizado no desenvolvimento do mesmo o mesmo sistema operacional.

5.3.2 Eclipse CDT

Um ambiente integrado para desenvolvimento muito conhecida é o eclipse. Existem inumeros *plugins* que facilitam o desenvolvimento de aplicações. O eclipse CDT utiliza-se destes plugins e apresenta mais uma vantagem, ser desenvolvido para programação na linguagem C++.

Algumas de suas facilidades no desenvolvimento para a linguagem C++ são:

- Indexação: O eclipse indexa *header files*, facilitando a função auto-completar.

- Configuração de *build*: É possivel desenvolver mais de um *makefile* e então definir qual compilação se deseja executar.

- *Debug* integrado com a IDE: Quando existe a necessidade de utilizar o *debug* da aplicação, o eclipse CDT já possui um *debug* integrado.

5.3.3 Linguagem de Programação C++

A utilização da linguaguem C++ é recomendada para este *framework* é recomendada, pois ambas as bibliotecas criptográficas,OpenSSL e Libcrypto-Sec, são desenvolvidas na lingaguem C e C++. Portanto, a conexão com estas bibliotecas se torna mais rápido e fácil, pois não necessita de um *middleware* para efetuar a conexão.

Outro ponto para o desenvolvimento em C++ é a robutez e flexibilidade da linguagem. Robustez no sentindo de velocidade na aplicação gerada e flexibilidade no ponto de que existem diversas bibliotecas e frameworks que podem ser utilizados.

Estudou-se a possibilidade de modificar a linguagem de programação, mas como os autores deste trabalho estão habituados com a linguagem C++, optou-se por permanecer nesta linguagem. Outro fator que contribuiu para esta escolha é a integração com bibliotecas criptográficas de alto desempenho citadas anteriormente.

5.3.4 Framework QT

> *"Qt é um conjunto de ferramentas de desenvolvimento de aplicações, gráficas, multi-plataforma que possibilita você compilar e executar suas aplicações em Windows, Mac OS X, Linux e diferentes marcas de Unix. Grande parte do Qt é dedicado a prover uma interface de plataforma neutra para tudo, desde a representação de caracteres na memória à criação de aplicações gráficas multi-tarefa."* (THELIN, 2007)

Segundo os criadores da ferramenta (Nokia e Trolltech), Qt é uma aplicação multi-plataforma e um *framework* UI ("*User Interface*" - Interface de usuário). Ele inclui uma biblioteca de classe multi-plataforma, ferramentas de desenvolvimento integrado e uma IDE ("*Integrated Development Environment*" - Ambiente de desenvolvimento integrado). Utilizando-se o Qt, é possível desenvolver aplicações uma vez utilizá-las em computadores e siste-

mas operacionais embarcados sem a ter a necessidade de reescrever o código fonte.

> *"Qt isola-nos o tanto quanto for possível de detalhes irrelevantes e complexidade subjacente para prover API's de fácil usabilidade que possamos usar de maneira simples e direta com grande efeito."* (SUMMERFIELD, 2010)

O uso desta ferramenta no desenvolvimento de aplicações auxilia diretamente no aumento da produtividade dos desenvolvedores. Processos repetitivos e/ou complexos são fornecidos na forma de serviços, tal como conexão ao banco de dados, são fornecidos por este *framework*. Fazendo com que o foco seja na parte criativa, na adição de valores à implementação.

5.3.5 TRAC

O TRAC é um sistema para o auxilio na gerência de projeto. Como neste trabalho é proposto o uso do *Scrum*, o TRAC auxilia o seu uso pois possui um *plugin* para desenvolvimento ágil.

Neste módulo de desenvolvimento ágil, o TRAC auxilia na criação de histórias, tarefas, *bugs* e melhorias. O fluxo para sua utilização funciona da seguinte forma: O Gerente cria histórias e em cada história ele cria tarefas, tarefas que desenvolvedor deve aceitar e implementar. Após a implementação desta tarefa ele deve repassar para o gerente de qualidade, o qual análisa o que foi implementado e se está dentro do padrão estipulado e finalmente a tarefa é finalizada ou repassada novamente para o desenvolvedor.

5.3.6 Bibliotecas Criptográficas

5.3.6.1 OpenSSL

O OpenSSL é um projeto *open source* para o desenvolvimento de protocolos *Secure Sockets Layer* (SSL) e *Transport Layer Security* (TLS).

O que implica na implementação de funções criptográficas usadas para o desenvolvimento de comunicação segura, além das implementações de algoritmos para criptografia que pode ser usado não só para a construção de comunicações seguras, mas para qualquer uso (OPENSSL, 2002).

A vantagem do uso da biblioteca OpenSSL são inúmeras, principalmente em ser bem consolidada e suas implementações seguirem o padrão do NIST (*National Institute of Standards and Technology*).

5.3.6.2 Libcryptosec

A biblioteca Libcryptosec já possui diversas classes,funções criptográficas que facilitam o desenvolvimento do projeto. Não necessitando o desenvolvimento por parte do framework dessas funções, uma vez que seguem algoritmos definidos.

As aplicações do LabSec já existentes em SGC e SAR, utilizam-se desta biblioteca para executar diversas funções. Como o framework tem como objetivo melhorar o funcionamento destas aplicações e manter o legado das mesmas é recomendavél por uma primeira instância manter o uso da biblioteca.

5.3.7 Servidor de Integração

O servidor de integração serve para verificar a integrinidade além de outras diretrizes definida na configuração do mesmo. O uso do *Jenkins*, que é uma ferramenta do âmbito de servidores de integração, é recomendada pois possui uma fácil instalação, existem diferentes *plugins* e possui documentação de fácil acesso.

5.3.8 Compilação Distribuida

A compilação distribuida vem como uma ferramenta para dar mais agilidade no processo de desenvolvimento. Tendo em vista que a compilação local, de projetos grandes, demora cerca de oito a nove minutos dependendo da configuração do computador.

Em testes efetuados dentro do LabSec, onde foi utilizado cerca de quatro servidores para a compilação, o tempo para a compilação de um projeto grande foi de três a cinco minutos.

5.4 CONFIGURAÇÃO DO AMBIENTE DE TRABALHO

A configuração do ambiente de trabalho define as ferramentas e tecnologias que devem ser usadas no desenvolvimento da solução. É essencial que todos desenvolvedores possuam um ambiente de trabalho semelhante, para garantir a compatibilidade entre os códigos gerados e para agilizar o processo de criação de novos ambientes de trabalho.

5.4.1 Máquinas de Desenvolvimento

O primeiro passo da configuração do ambiente é a definição de qual sistema operacional os desenvolvedores devem usar em suas máquinas. Devido ao legado dos projetos anteriores, e por ser um requisito da principal parceria do LabSEC, o sistema operacional deve ser o Red Hat Enterprise Linux (RHEL), na sua versão 6. O desenvolvimento deve ser feito na linguagem C++, com uso do framework QT, na sua versão 4. Para as funções criptográficas, deve ser utilizada a biblioteca *libcrypto*, do OpenSSL, e *libcryptosec*, do LabSEC. O projeto deve possuir controle de versão, que pode ser feito através da ferramenta *SVN*. Para a implementação de código, a IDE *Eclipse CDT* deve ser utilizada, em conjunto com o *plugin Subclipse*, que facilita o uso do controle de versão do *SVN*. Devido ao tamanho do projeto,

é necessário o uso de compiladores distribuidos, que reduzem o tempo de compilação, para isso deve ser usado a feramenta *Distcc*, que deve estar configurada, como cliente e servidor, em todas as máquinas do projeto.

5.4.2 Servidores

5.5 SEGMENTAÇÃO DE DESENVOLVIMENTO

O projeto da LibPKi foi dividido em três frentes de desenvolvimento: persistência, modelo e interface gráfica. A intenção é que esses três módulos sejam desenvolvidos separadamente, por equipes diferentes, ou pela mesma equipe, em etapas diferentes do desenvolvimento. O ponto de conexão entre eles são as interfaces especificadas na modelagem dos componentes e pelas estruturas de dados, que trafegam entre essas interfaces. A seções a seguir especificam as funções de cada módulo.

5.5.1 Persistência

O módulo de persistência de dados deve prover componentes que implementem as funções das interfaces de salvamento e recuperação de dados, requeridas pelos outros módulos. A implementação dos componentes deve trabalhar com a linguagem SQL, e deve se manter independente da distribuição do banco de dados. O desenvolvimento desses componentes deve utilizar as classes do *sub-framework QtSql*, do *QT*. Além dos componentes de salvamento e recuperação de dados, esse módulo deve prover:

- Classes de abstração de conexão com o banco de dados;
- Classes para integridade de banco de dados;
- Classes para geração e recuperação de backups;
- Componentes para salvamento e recuperação de dados em arquivos.

5.5.2 Modelo

As classes e componentes do módulo de modelo são dividios em duas categorias, criptografia e infraestrutra de chaves públicas. A primeira tem o conjunto das classes e componentes que são responsáveis pelas operações criptográficas; a segunda tem o conjunto de classes e componentes que representam entidades da ICP, como autoridades certificadoras e autoridades registradoras.

Esse módulo não deve utilizar o framework QT, e deve se manter independente dos padrões estabelecidos pelas bibliotecas criptográficas *libcrypto* e *libcryptosec*, encapsulando-as em classes com interfaces bem definidas.

5.5.3 Interface Gráfica

O desenvolvimento do módulo de interface gráfica deve trabalhar com as ferramentas e classes fornecidas pelo *framework QT*. Uma dessas ferramentas é o *QtDesigner*, que facilita a criação de componentes de interface gráfica.

Os componentes desenvolvidos nesse módulo podem ser simples, como pequenos formulários, ou complexos, como um wizard de criação de certificado, formado por diversos componentes de formulário. Além dos componentes de interface, devem existir, nesse módulo, classes para validação e controle de dados.

5.6 PRODUÇÃO DE DOCUMENTAÇÃO

Junto com o processo de desenvolvimento deverá ser executado em paralelo a produção de uma documentação. Como este *software* tem como objetivo final os desenvolvedores de aplicação, a documentação gerada é para o auxilio no desenvolvimento.

Nesta documentação deverá conter, as classes, os métodos providos

pelo *framework* assim como o retorno destes métodos. Deverá conter também exemplos de uso de cada classe, auxiliando o desenvolvedor a entender melhor as funções.

5.7 ETAPAS DE DESENVOLVIMENTO

5.7.1 Prototipação de Software

O termo prototipagem designa um conjunto de técnicas utilizadas para propor e implementar objetos concretos derivados do planejamento de um projeto. A intenção ao se desenvolver um protótipo é possuir um modelo que dê auxílio visual e prático das ideias sendo expostas e que permitam a evolução do projeto. O processo de desenvolvimeto de um protótipo pode ser visto como uma divisão de etapas bem definidas, como mostra a figura 18.

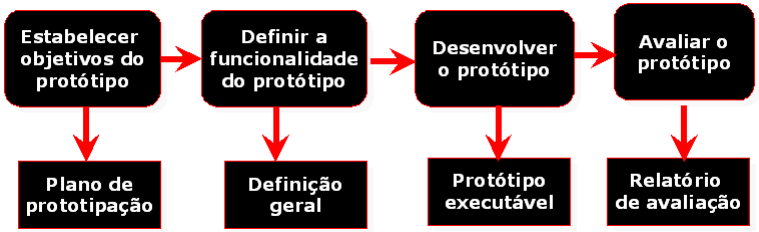


Figura 28 – "Processo de prototipação de *software* - fonte: (PRESSMAN, 2006) Modificado pelos autores"

Um protótipo pode ser categorizado de diversas formas, tais como incremental, evolutivo, extremo ou de descarte. Neste trabalho faremos uma abordagem de descarte, pois o protótipo é uma simples representação da ideia geral e por isso não abordaremos diversos pontos importantes necessários ao funcionamento completo e preciso. Representaremos no entanto um protótipo que exibe um fluxo básico e geral de funções básicas no contexto de ICP.

O objetivo do protótipo sendo desenvolvido é fazer uma validação dos requisitos já expostos e exibir o funcionamento prático do planejamento

acerca de frameworks e ICP's, assim como identificar possíveis erros no modelo ou funcionalidades esquecidas.

5.7.2 Implementação das Interfaces dos Componentes

Conforme descrito na seção de modelagem, existem interfaces que os componentes provém que devem ser implementados. Não a interface em si, mas sim os metodos que esta chama.

5.7.3 Implementação das Estruturas de Dados

As estruturas de dados que o *framework* proporciona, boa parte delas estão implementadas na libcryptosec. Contudo, as estruturas de dados de conversasão entre persistência, model e interface precisam ser implementados.

5.7.4 Implementação dos Componentes

Conforme foi descrito na seção de Modelagem, o *framework* é composto por diversos componentes. Estes componentes precisam ser implementados de acordo com os diagramas de atividades que foram propostos neste trabalho.

Esta implementação de componentes, deve obedecer os requisitos que foram propostos. Tentando englobar o maior casos de uso e deixando o *framework* o mais abstrato possivel, cuidando para que o mesmo não fique engessado.

5.8 PROCESSO DE DESENVOLVIMENTO

Conforme descrito na fundamentação teórica deste trabalho, durante o processo de desenvolvimento utilizamos diferentes técnicas para que o *software* conseguisse ser confeccionado de forma eficiente e eficaz.

Primeiramente no processo de desenvolvimento, foi criado as histórias de uso conforme o método ágil *scrum* determina. Nesta etapa, além das histórias de uso outros artefatos como, as tarefas foram criadas.

Conforme as tarefas eram geradas as *sprints* eram definidas também. E a medida que eram desenvolvidas as classes do *framework*, a geração dos testes unitários conforme o *TDD* eram criadas também, garantindo assim a eficácia do *framework*.

As primeiras tarefas geradas, foram o desenvolvimento da modelagem discutida na seção Modelagem deste trabalho. Logo após a modelagem ser definida, em seguida começou-se a desenvolver código própriamente dito. Nesta etapa após a modelagem, etapa de desenvolvimento, foi-se desenvolvido as interfaces gráficas, classes abstratas e concretas, e interfaces.

Os autores deste trabalho recomentam que seja dividido para que tenha uma maior cobertura no desenvolvimento do *framework*.

O que consta na parte de protótipo é a criação e implementações de classes e interfaces. Como já foi especificado neste trabalho, o artefato final de todo o desenvolvimento é um *framework*. Sendo assim, o produto final é um *software*, um programa completo, contudo não é o *software* que estamos acostumados após o final de todo o processo de desenvolvimento.

Conforme demonstrado na seção de Modelagem, foi gerado diagramas e outros aterfatos de engenharia de software. Com as técnicas descritas na seção de desenvolvimento, finalizou-se o *framework* com a geração das classes e interfaces de cada componente.

Deverá ser detalhado nesta seção o motivo pelo qual foi escolhido está forma de geração de classes. Os autores, conforme pesquisado, desenvolveram o *framework* para que este consiga ter uma abrangência maior para a resolução de problemas em ICP.

6 CONSIDERAÇÕES FINAIS

O desenvolvimento de um *framework* por parte dos autores deste trabalho, contribuiu no auxílio no processo de desenvolvimento de diversas aplicações no âmbito de Infraestrutura de chaves públicas. Além, é claro, do desenvolvimento pedagógico e intelectual dos autores.

Entre os resultados deste trabalho está a capacidade do *framework* de gerar um Sistema Gerenciador de Certificados. Através da união dos componentes especificados, pode-se gerar uma aplicação completa, apenas efetuando a ligação entre os componentes.

O conhecimento durante o processo de desenvolvimento de toda LibPKI, proporcionou um aprendizado de grande valor. O uso de novas técnicas de modelagem, como a modelagem por componentes, a utilização do método *scrum* assim como o uso de integração continua e desenvolvimento orientado à testes agregou valores aos autores quanto ao *software*.

Pode-se destacar o auxílio de diversas fontes de conhecimento, como o prof. Dr. Ricardo Pereira entre outros, para a qualidade do *framework*. O conhecimento sobre o domínio do problema e sobre a linguagem C++ foi essêncial para desevolver o projeto.

6.1 TRABALHOS FUTUROS

Atualizações continuas ao *framework* para que não caia em desuso e não se torne obsoleto, como a adição de novos algoritmos criptográficos, melhorias de usabilidade na interface gráfica e o suporte a mais bancos de dados. Propõe-se a implementação de suporte *online*, assim extendendo o suporte do *framework* para aplicações no âmbito de infraestrutura de chaves públicas. Com este suporte implementado, é necessário a adequeção do *software* para que seja *Thread-Safe*, ou seja para que funcione corretamente tendo em vista que diferentes *threads* estarão acessando algoritmos criptográficos e dados do

banco de dados.

Outro trabalho futuro proposto é a adição ao suporte de MSC, pois existem diferentes modelos de MSC atualmente. Visando flexibilizar o uso deste *software*, a adição de suporte a mais de um tipo de MSC, ou pelo menos classes que sejam extendiveis para que o programador possa desenvolver *drivers* e utilizar o MSC desejado.

6.2 ANEXOS

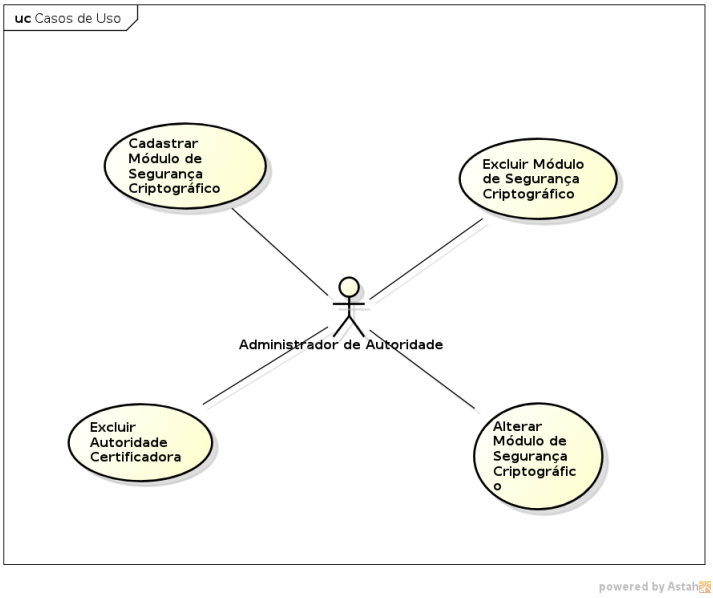


Figura 29 – “Casos de uso do *framework* para Administração.”

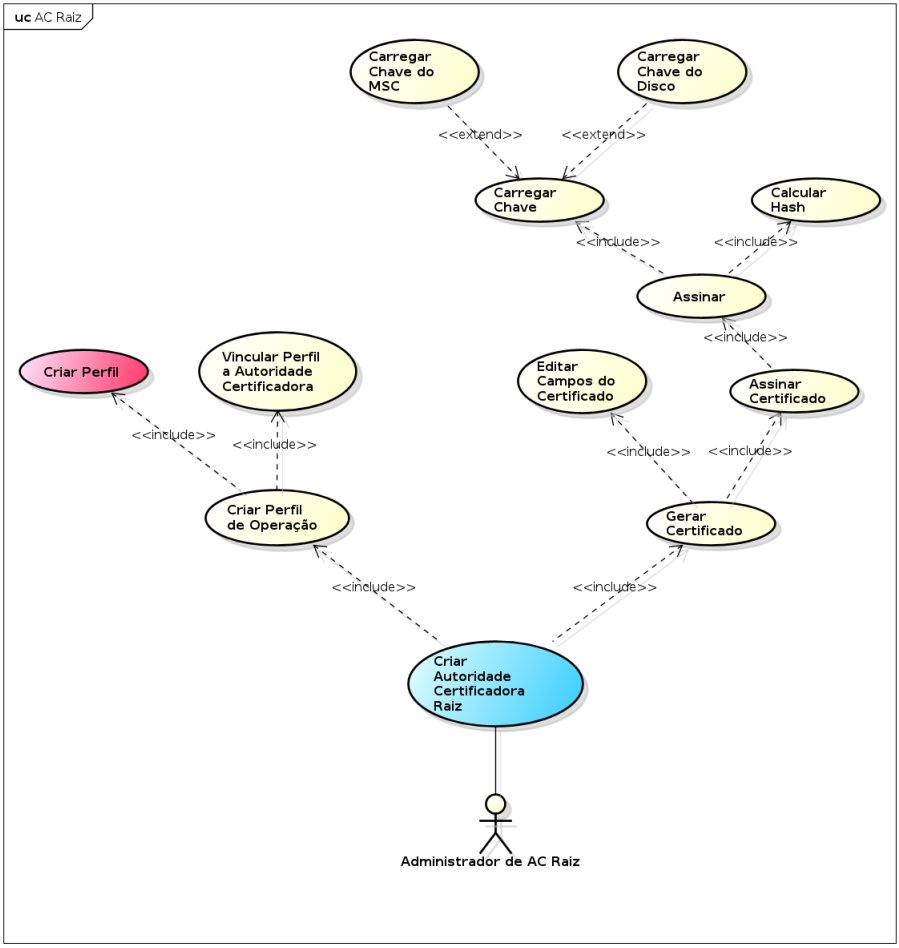


Figura 30 – “Casos de uso do *framework* para Administração de uma AC Raiz.”

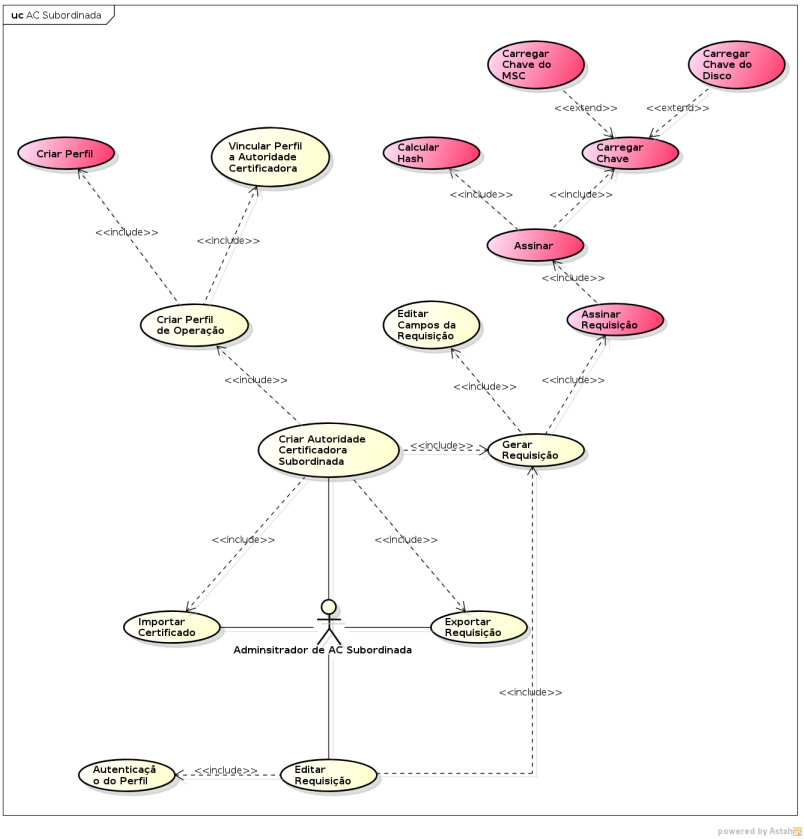


Figura 31 – “Casos de uso do *framework* para Administração de uma AC Subordinada.”

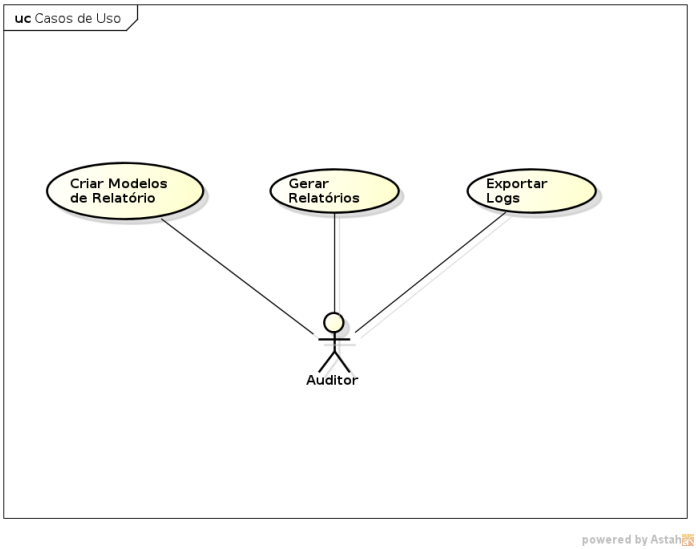


Figura 32 – “Casos de uso do *framework* para Auditoria.”

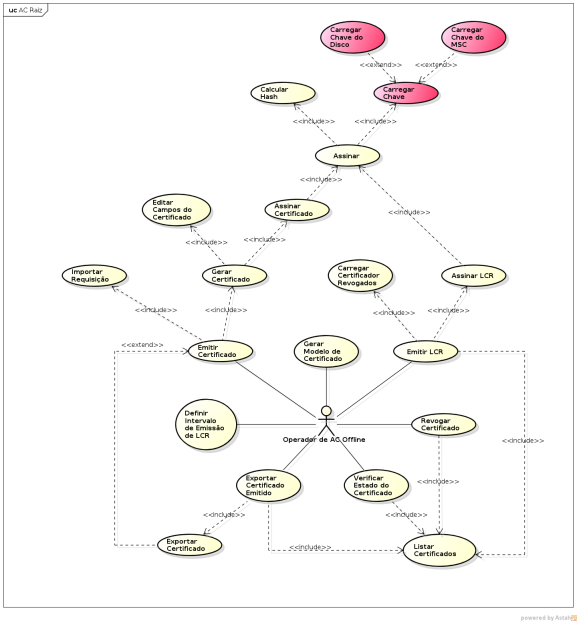


Figura 33 – “Casos de uso do *framework* para Operação de uma AC Raiz.”

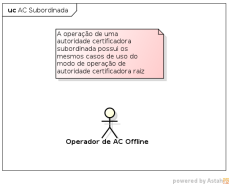


Figura 34 – “Casos de uso do *framework* para Operação de uma AC Subordinada.”

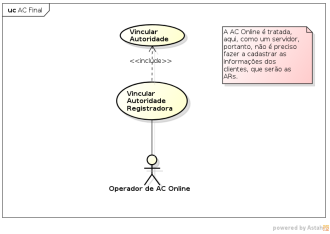


Figura 35 – “Casos de uso do *framework* para Operação de uma AC Final.”

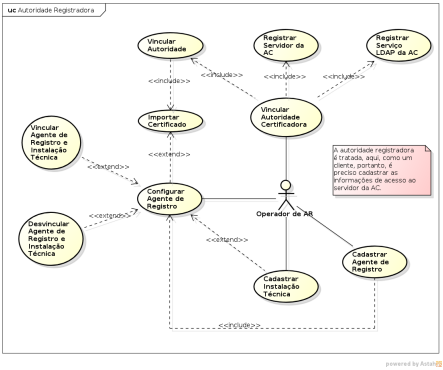


Figura 36 – “Casos de uso do *framework* para Operação de uma AR.”

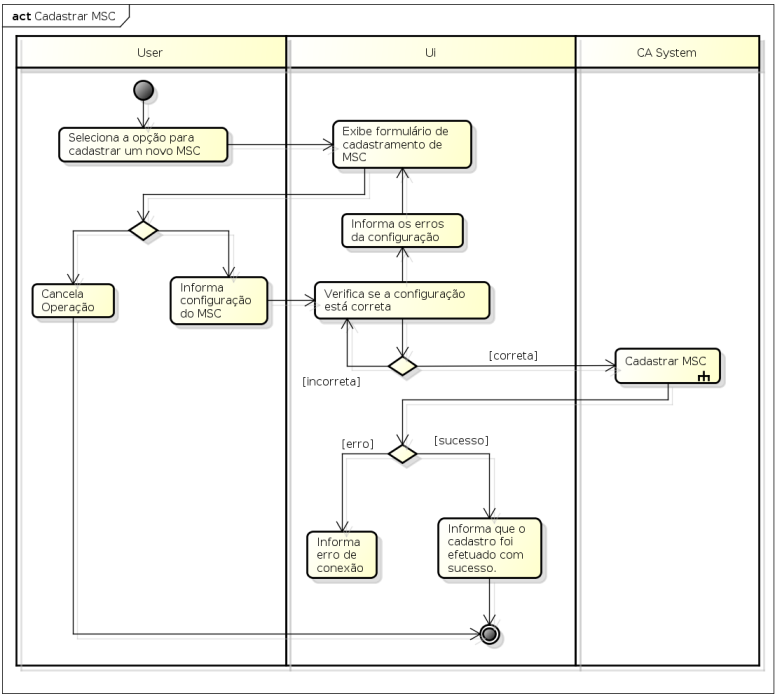


Figura 37 – “Figura referente ao cadastro de MSC.”

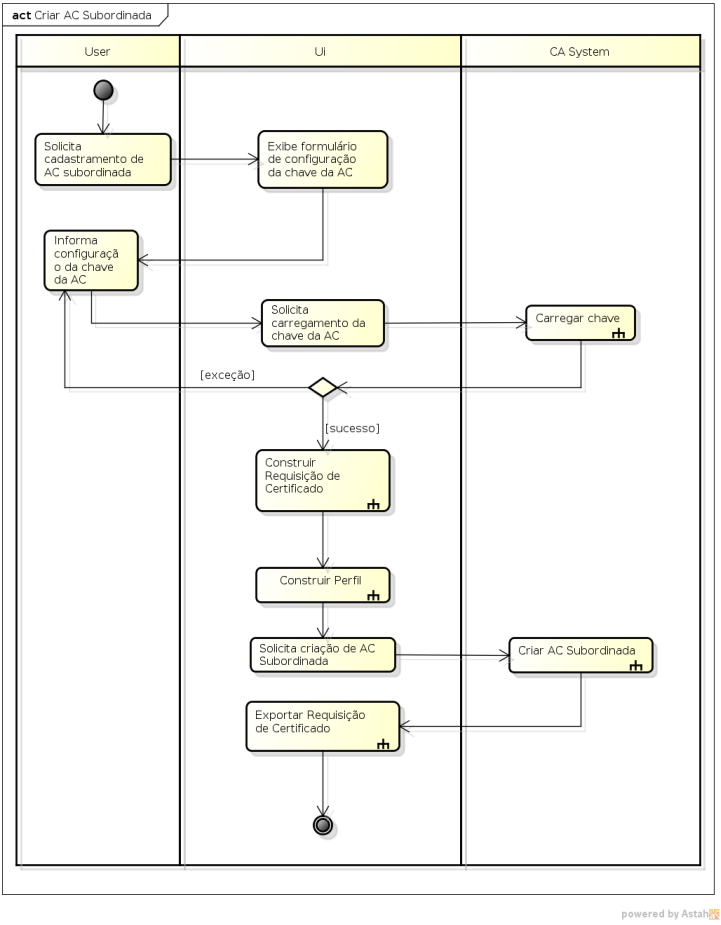


Figura 38 – “Figura referente a criação de uma AC Subordinada.”

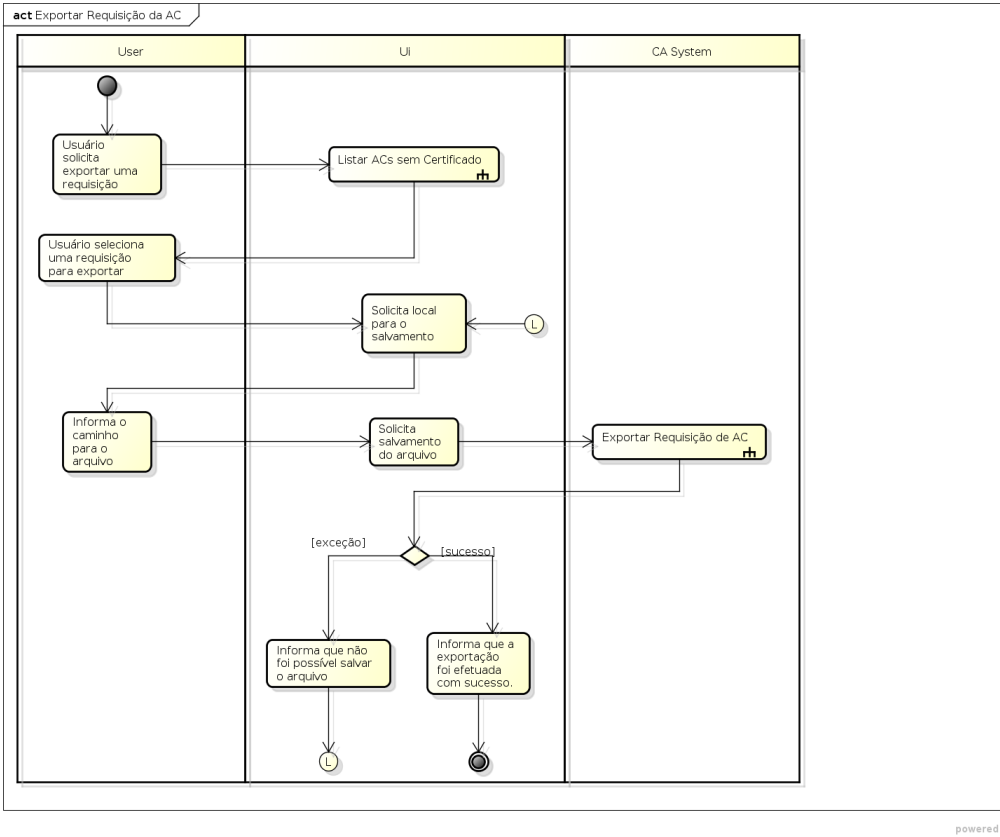


Figura 39 – “Figura referente a exportação de uma Requisição.”

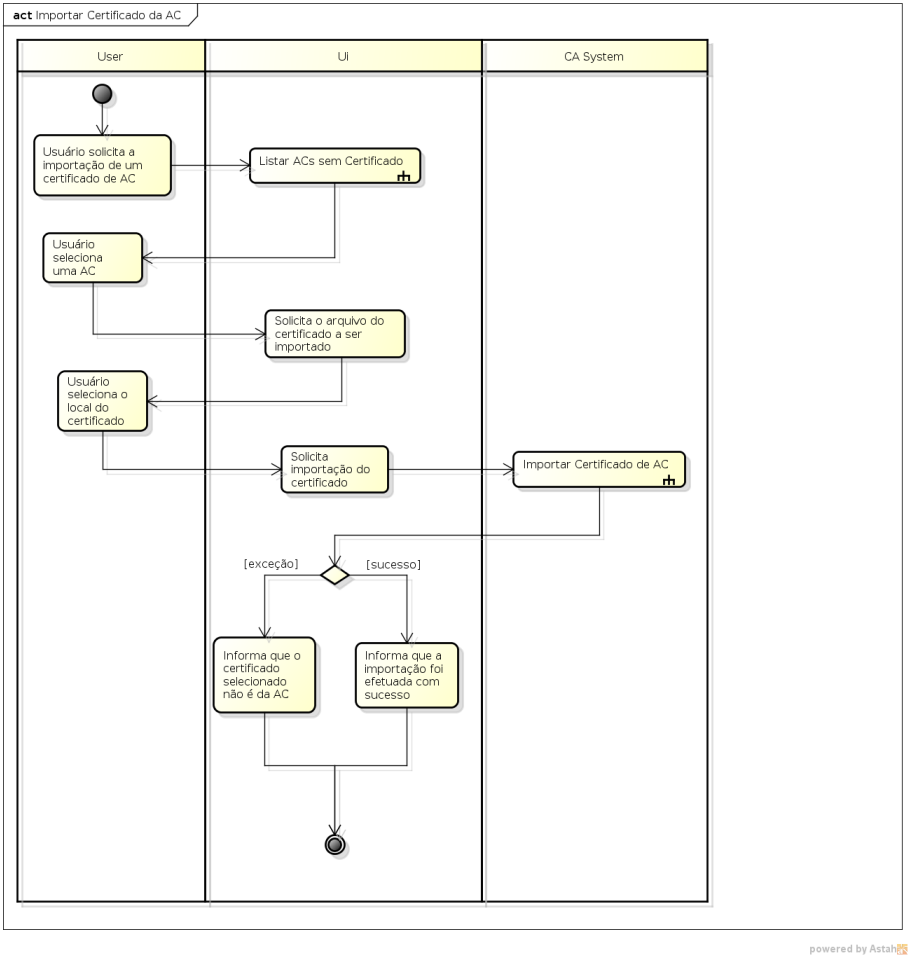


Figura 40 – “Figura referente a importar o certificado da AC Subordinada ou AC Final.”

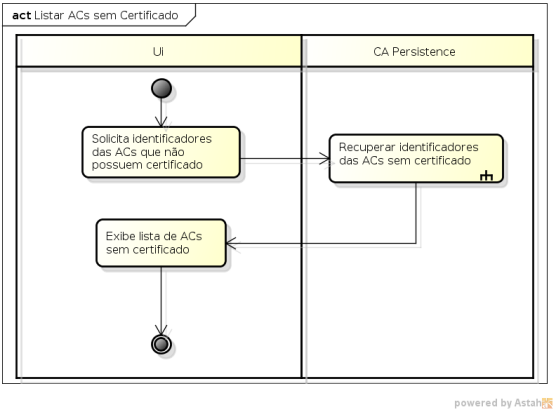


Figura 41 – “Figura referente a listar ACs sem certificado.”

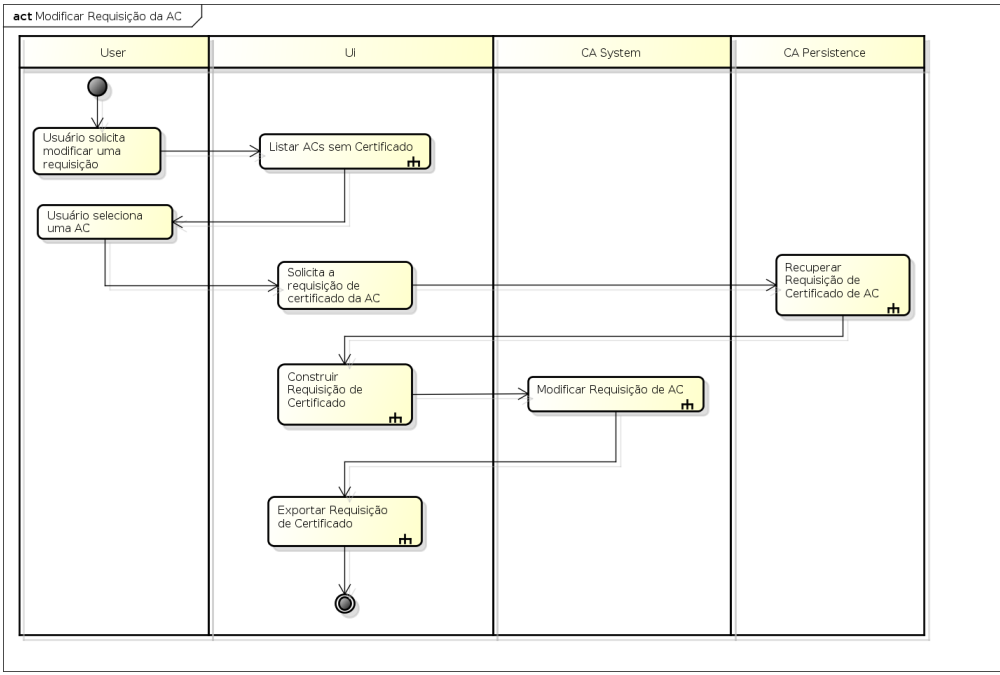


Figura 42 – “Figura referente a modificação de uma requisição de AC.”

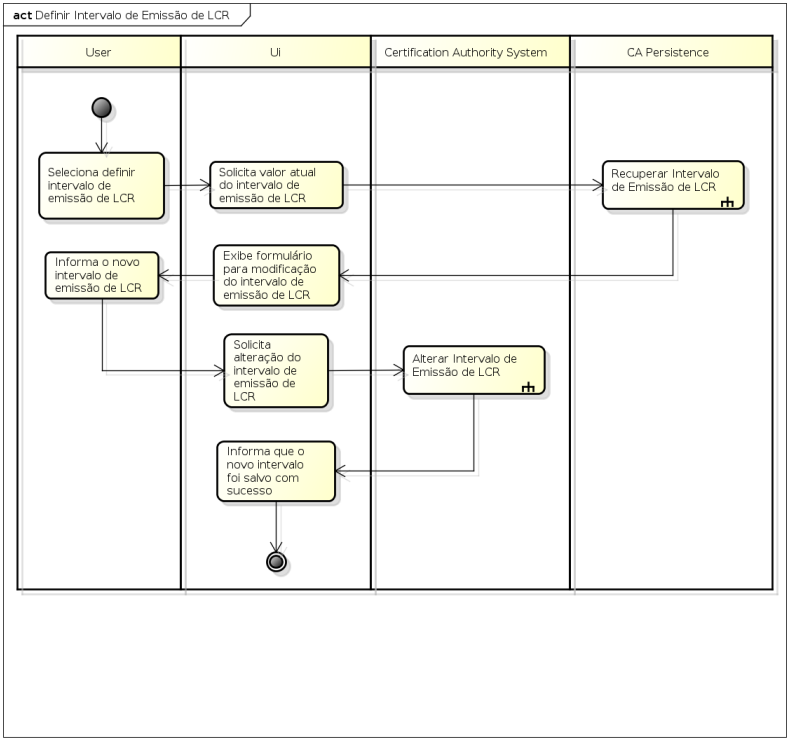


Figura 43 – “Figura referente a definição de intervalo de emissão de LCR.”

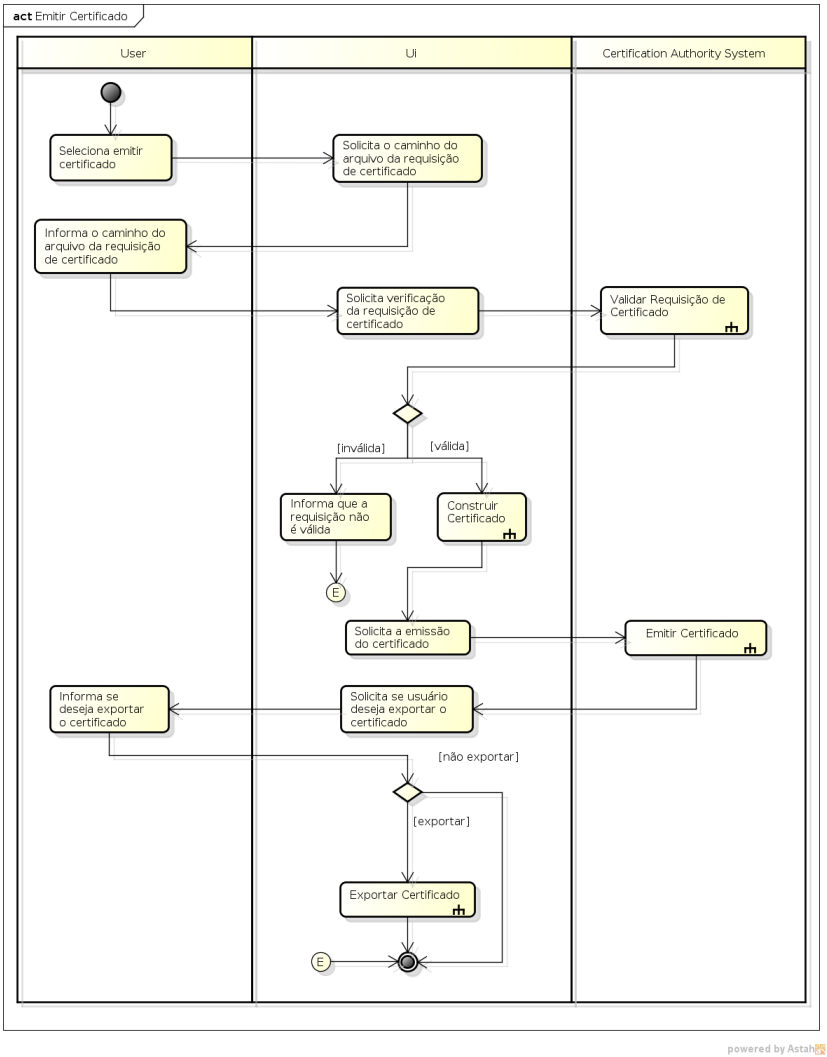


Figura 44 – “Figura referente a emissão de um Certificado.”

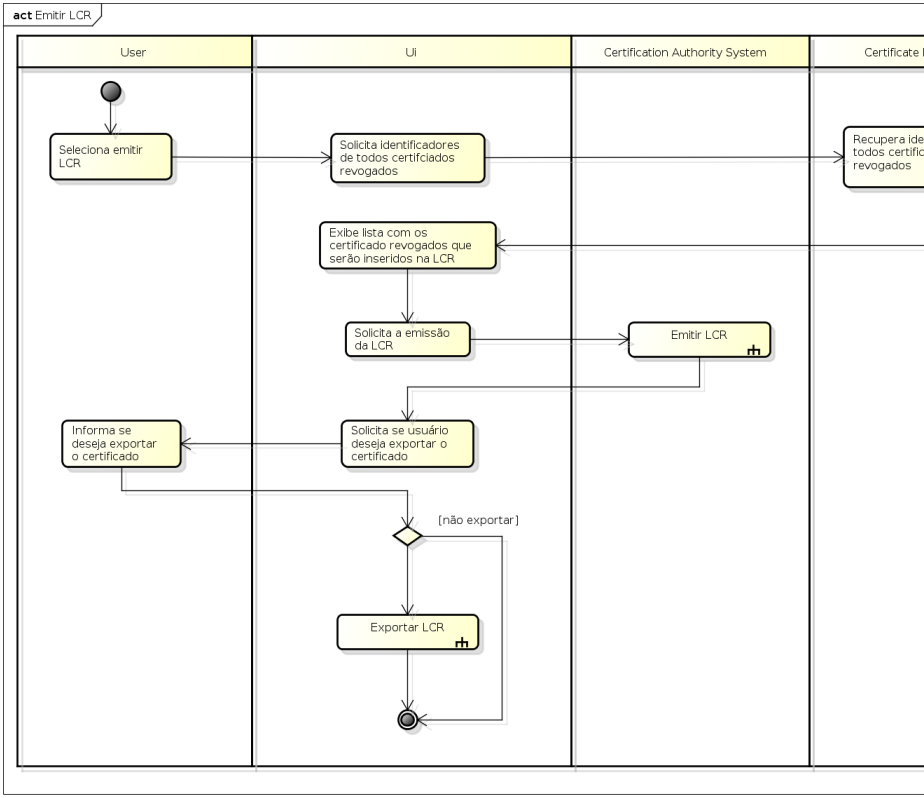


Figura 45 – “Figura referente a emissão de uma LCR.”

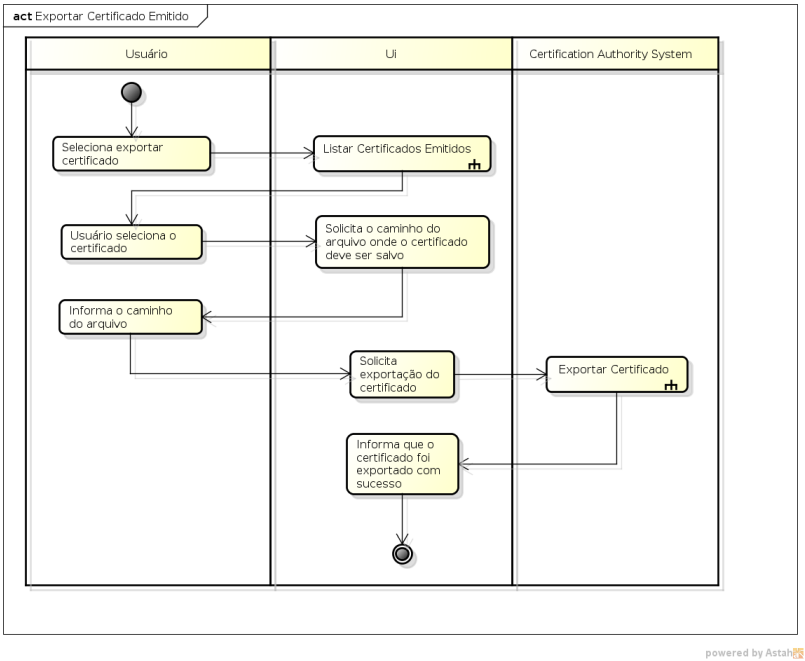


Figura 46 – “Figura referente a exportação de um certificado emitido.”

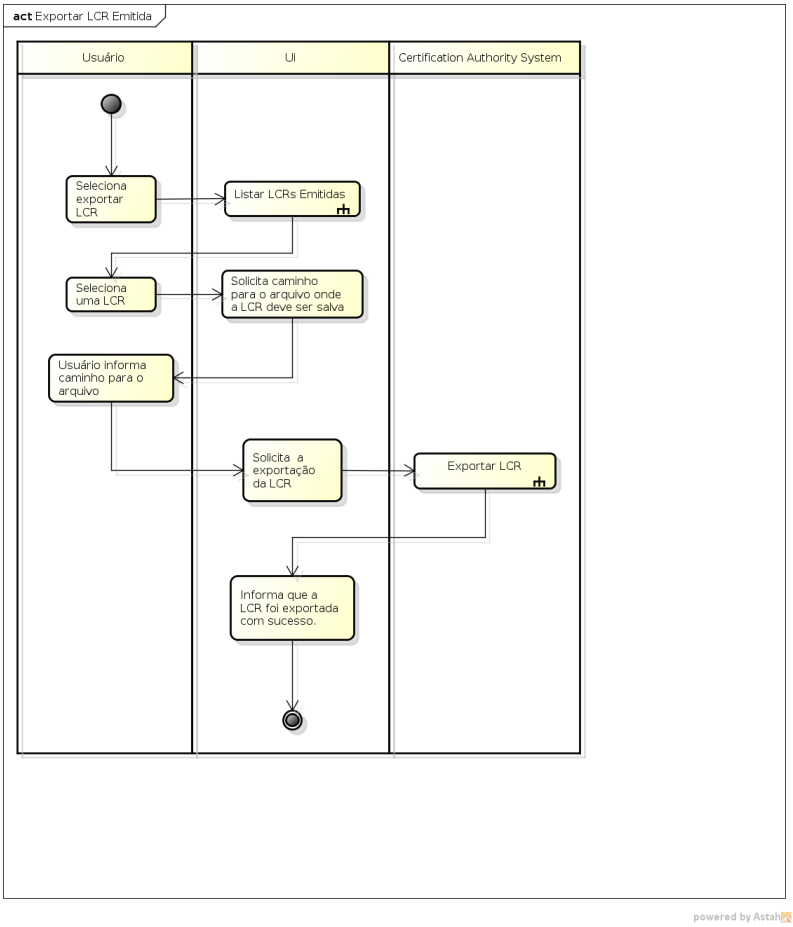


Figura 47 – “Figura referente a exportação de uma LCR emitida.”

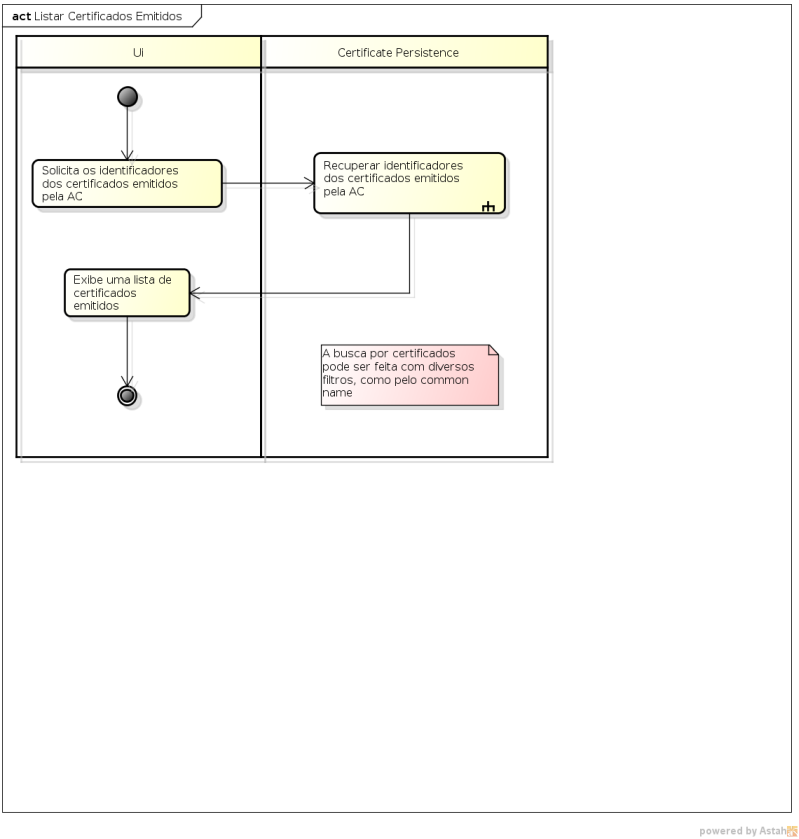


Figura 48 – “Figura referente ao diagrama para listar certificados emitidos.”

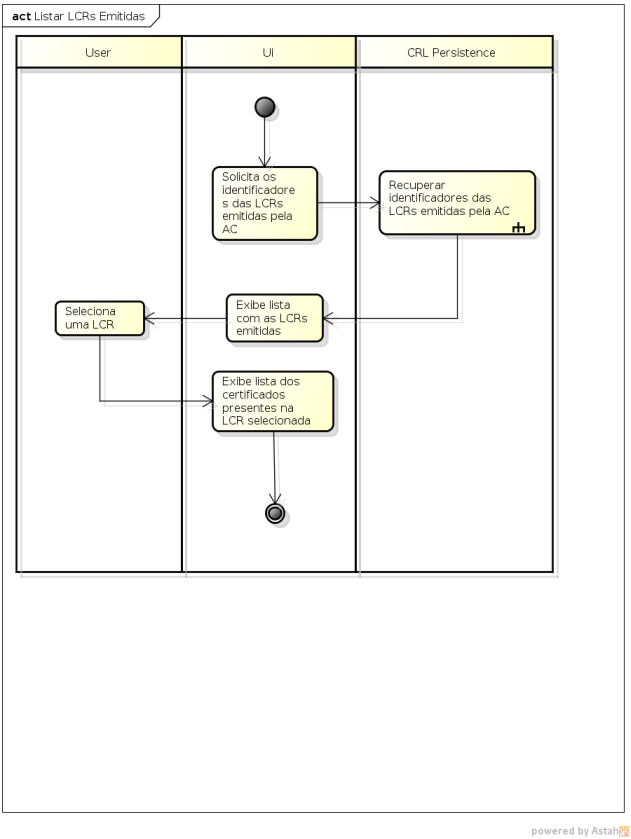


Figura 49 – “Figura referente ao diagrama para listar LCRs emitidas.”

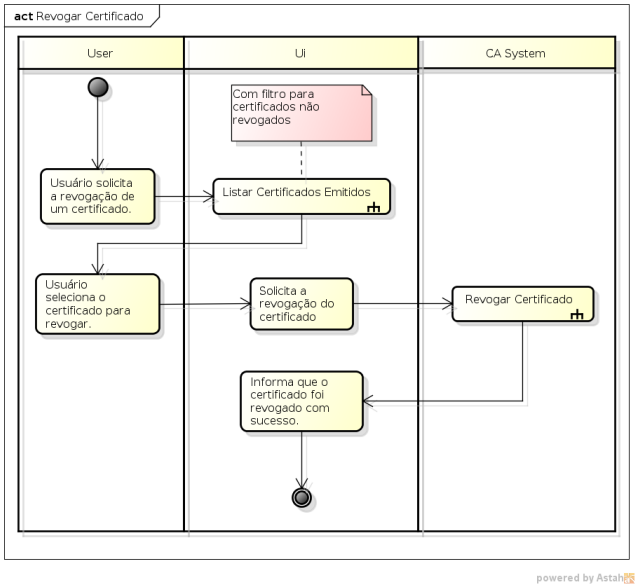


Figura 50 – “Figura referente a revogação de um Certificado.”

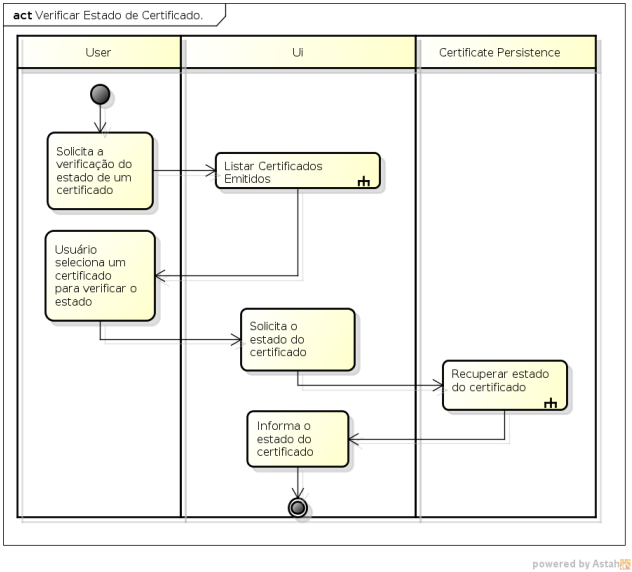


Figura 51 – “Figura referente a verificação do estado de um certificado.”

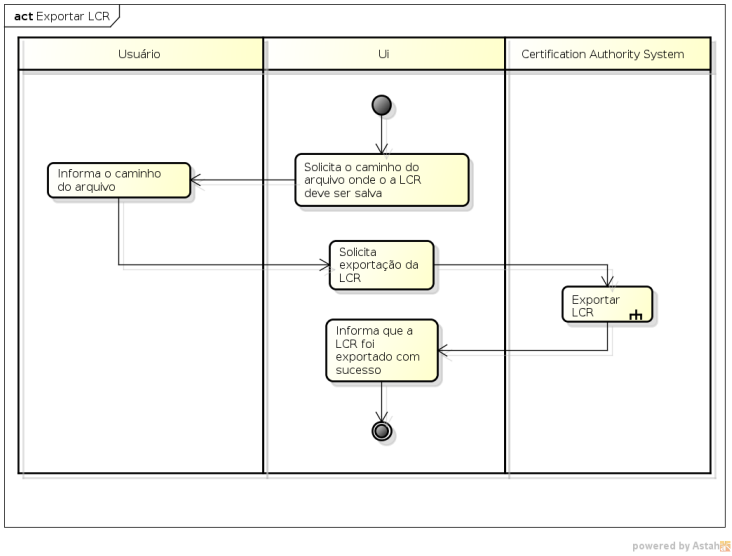


Figura 52 – “Figura referente a exportar uma LCR.”

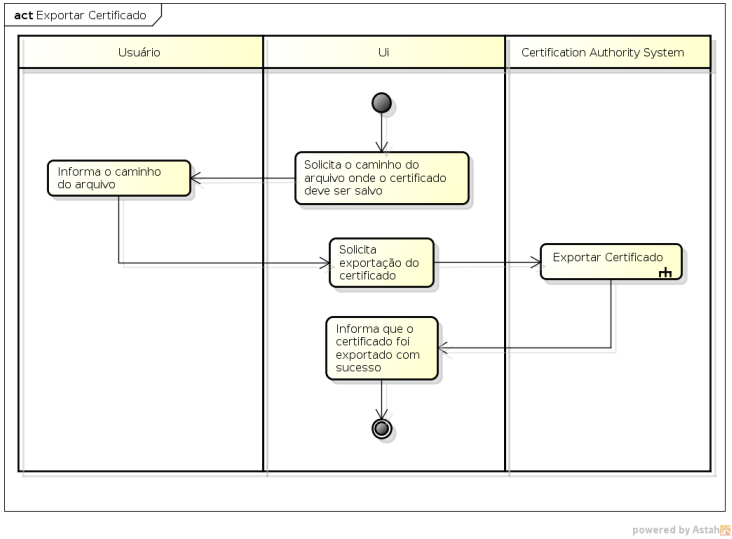


Figura 53 – “Figura referente a exportar um certificado.”

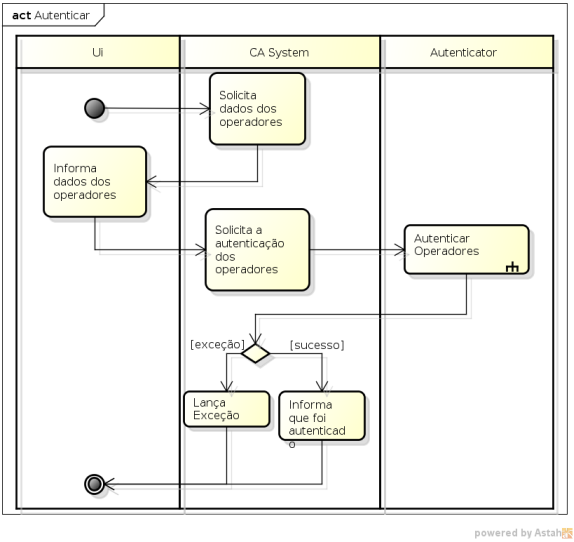


Figura 54 – “Figura referente ao diagrama para autenticação no sitema.”

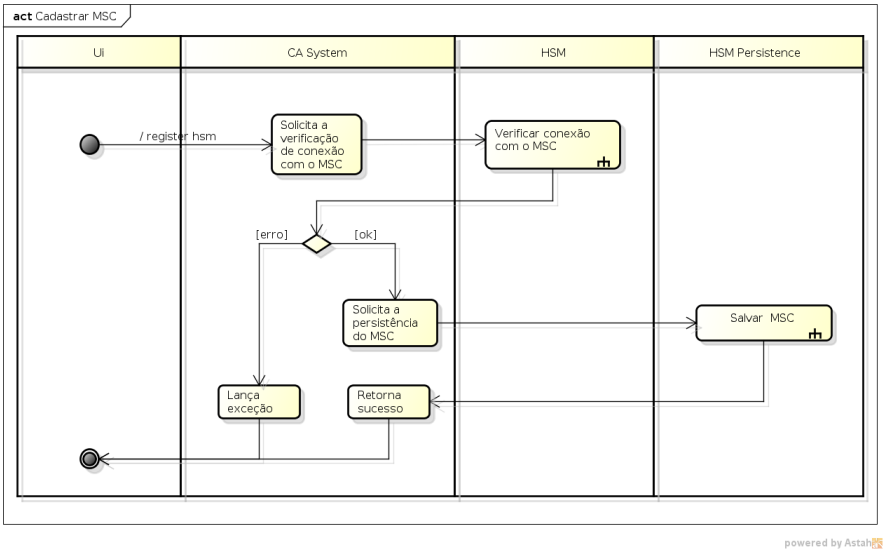


Figura 55 – “Figura referente a cadastrar um MSC na visão do sistema.”

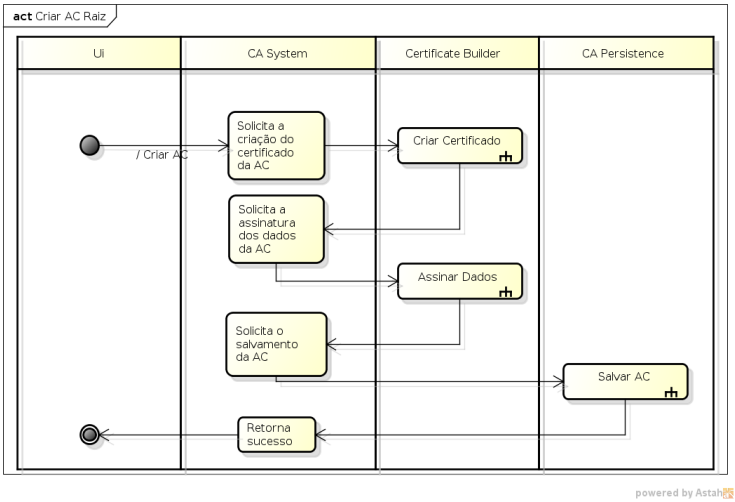


Figura 56 – “Figura referente a criação de AC Raiz na visão do sistema.”

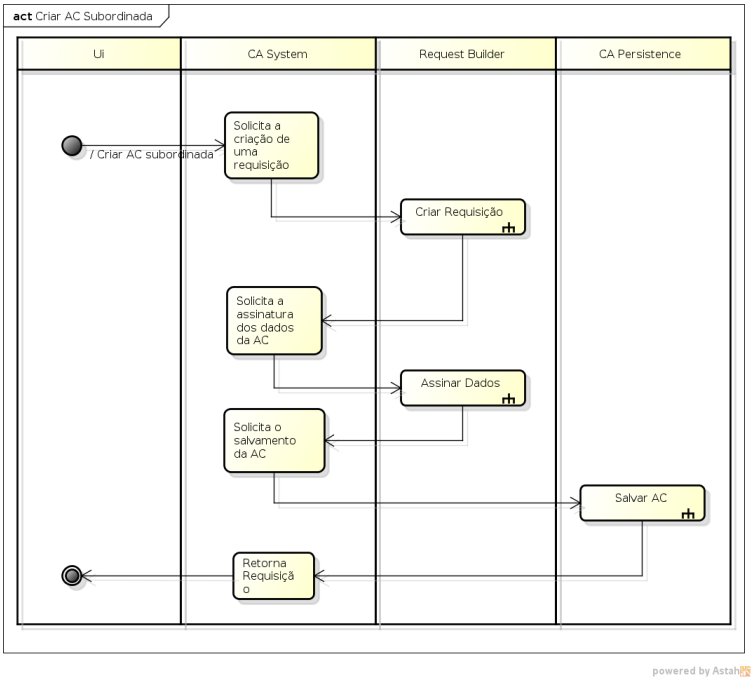


Figura 57 – “Figura referente a criação de AC Subordinada na visão do sistema.”

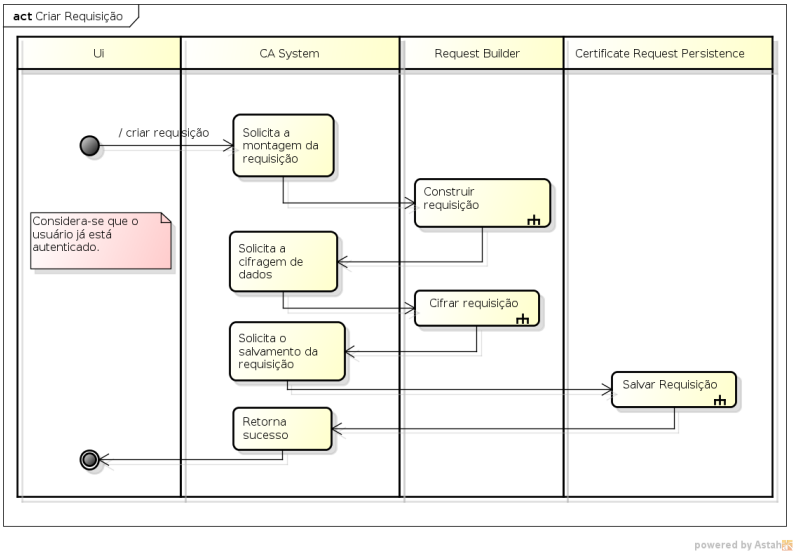


Figura 58 – “Figura referente a criação de uma requisição na visão do sistema.”

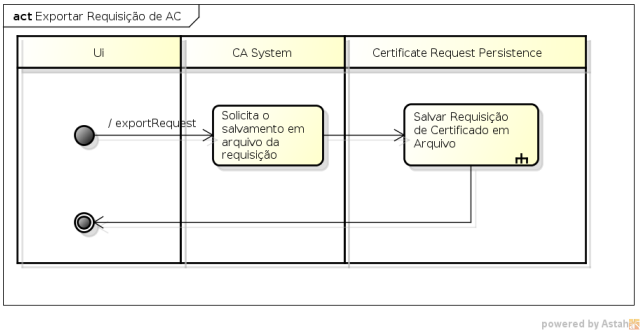


Figura 59 – “Figura referente a exportar uma requisição na visão do sistema.”

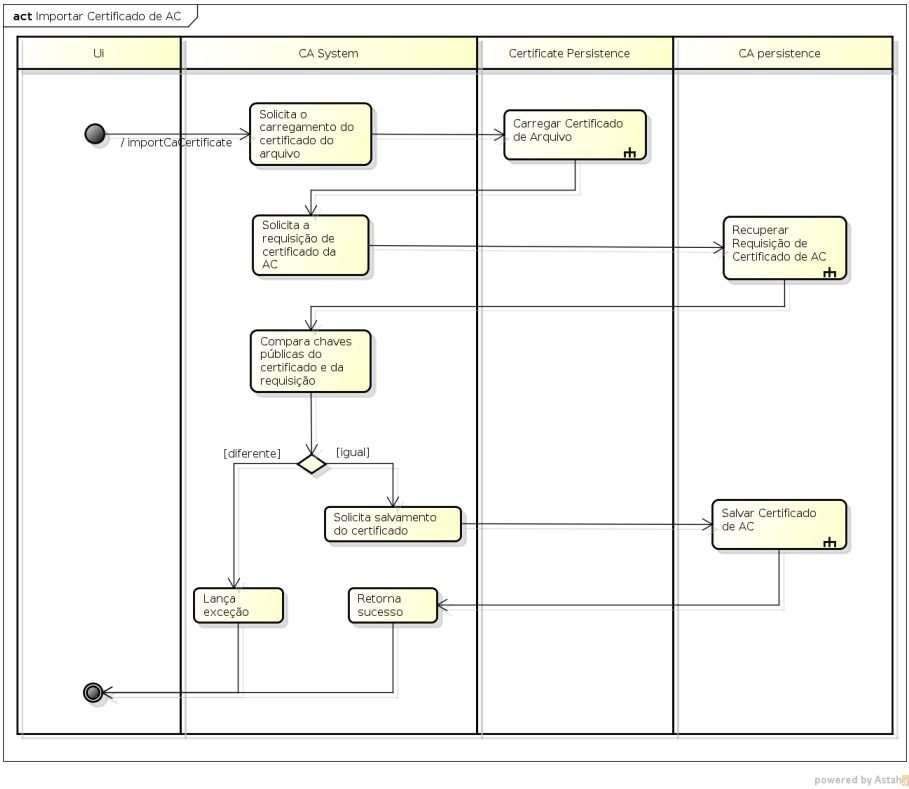


Figura 60 – “Figura referente a importar um certificado na visão do sistema.”

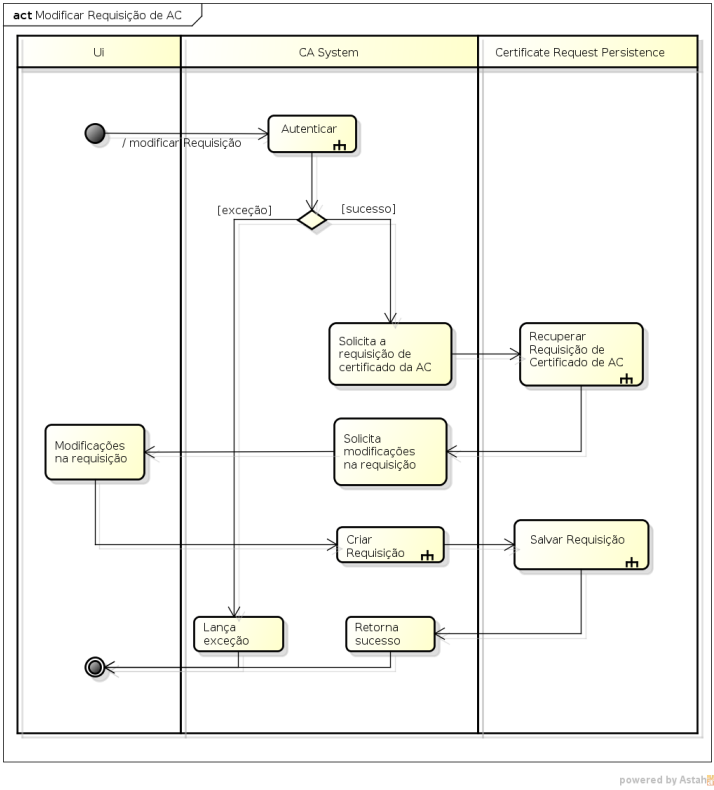


Figura 61 – “Figura referente a modificar uma requisição na visão do sistema.”

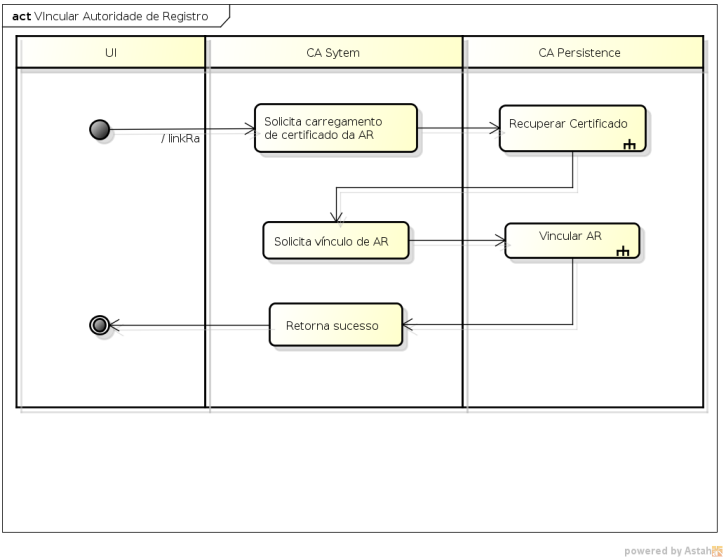


Figura 62 – “Figura referente a vincular uma Autoridade de Regitro na visão do sistema.”

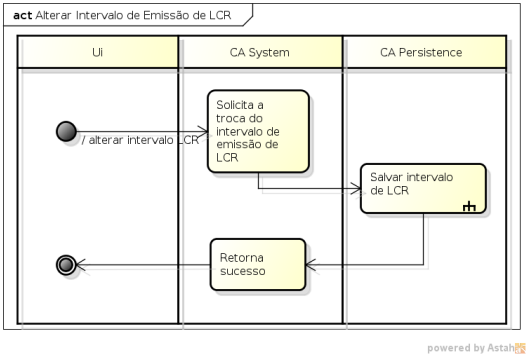


Figura 63 – “Figura referente a alteração do intervalo de emissão de uma LCR na visão do sistema.”

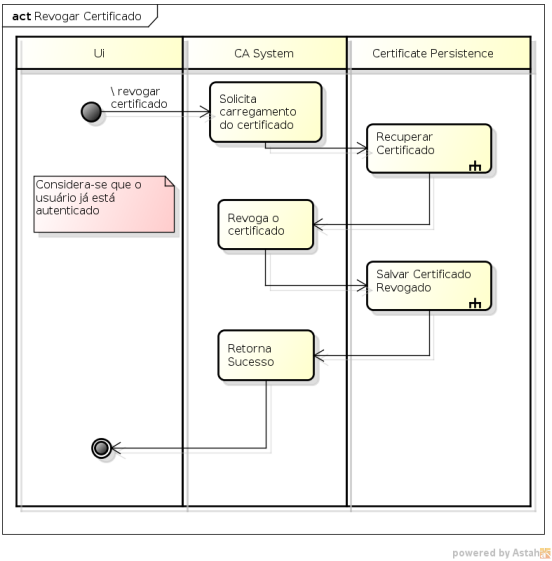


Figura 64 – “Figura referente a revogação do certificado na visão do sistema.”

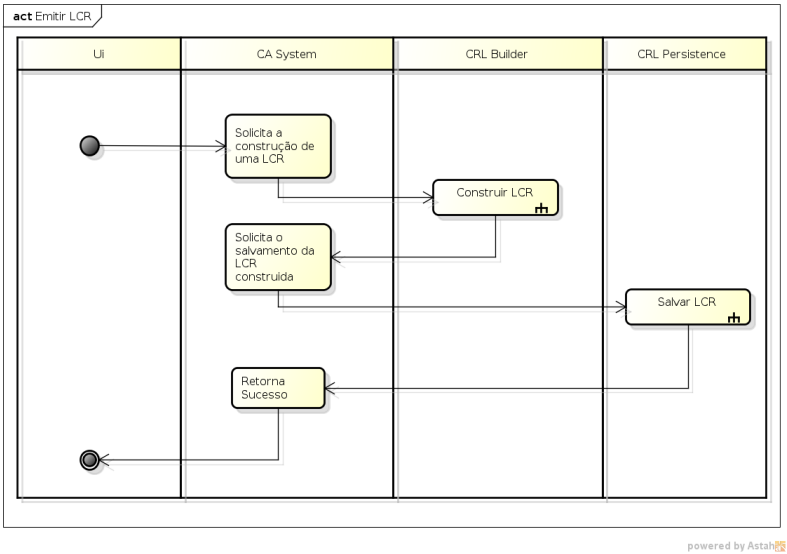


Figura 65 – “Figura referente a emissão de uma LCR na visão do sistema.”

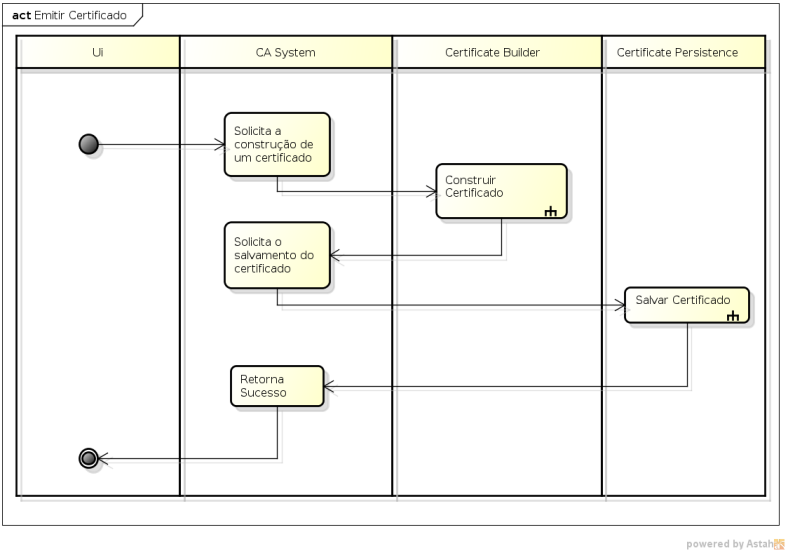


Figura 66 – “Figura referente a emissão de um certificado na visão do sistema.”

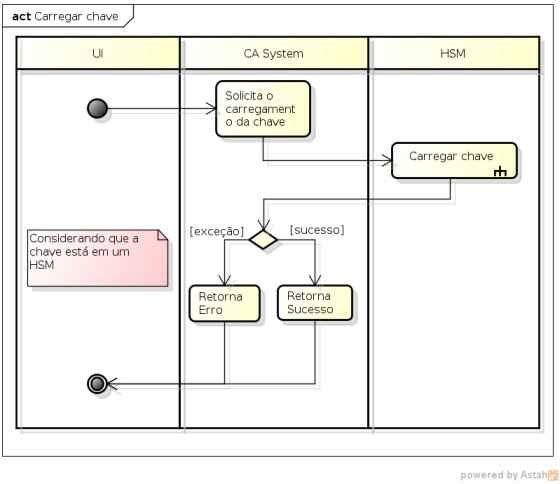


Figura 67 – “Figura referente ao carregamento de chaves.”

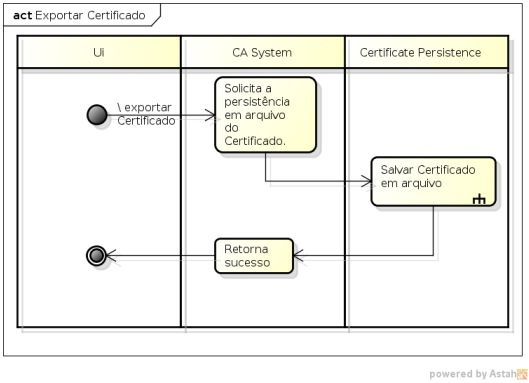


Figura 68 – “Figura referente a exportação de um certificado na visão do sistema.”

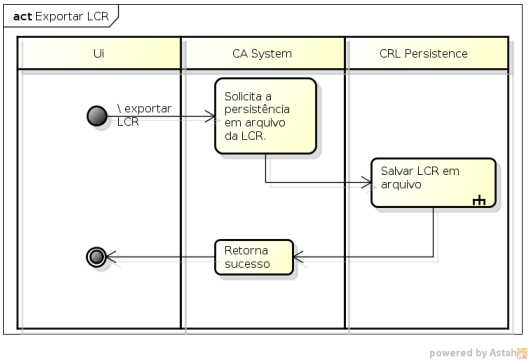


Figura 69 – “Figura referente a exportação de uma LCR na visão do sistema.”

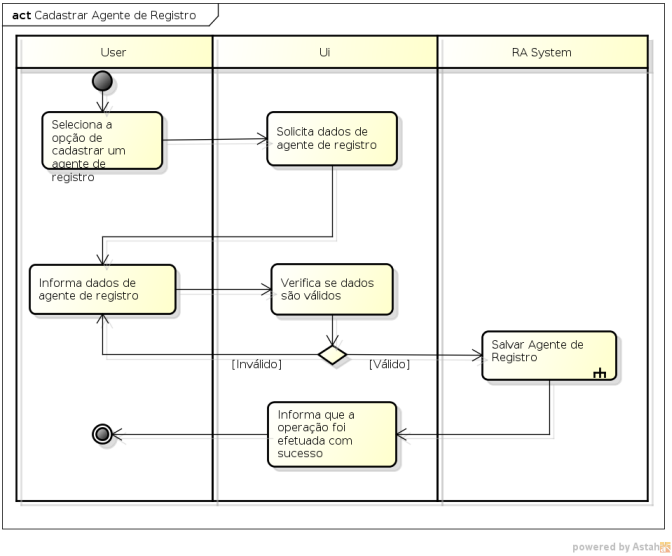


Figura 70 – “Figura referente ao cadastro de um Agente de Registro.”

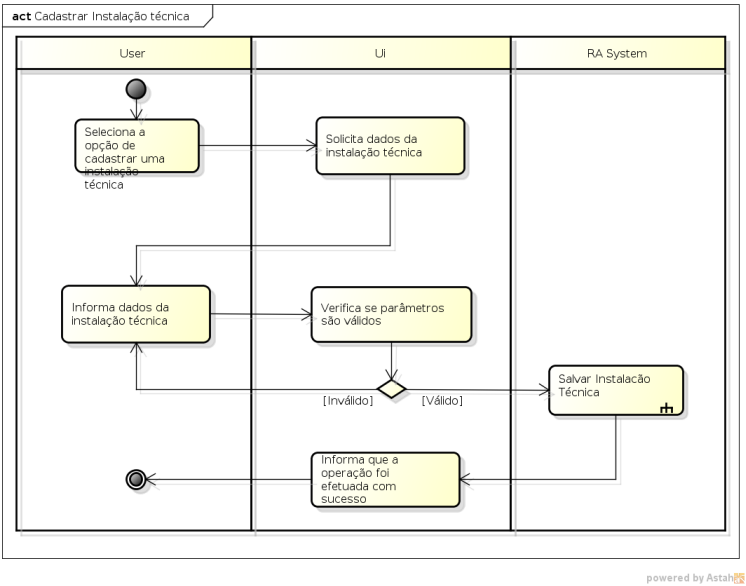


Figura 71 – “Figura referente ao cadastro de uma Instalação Técnica.”

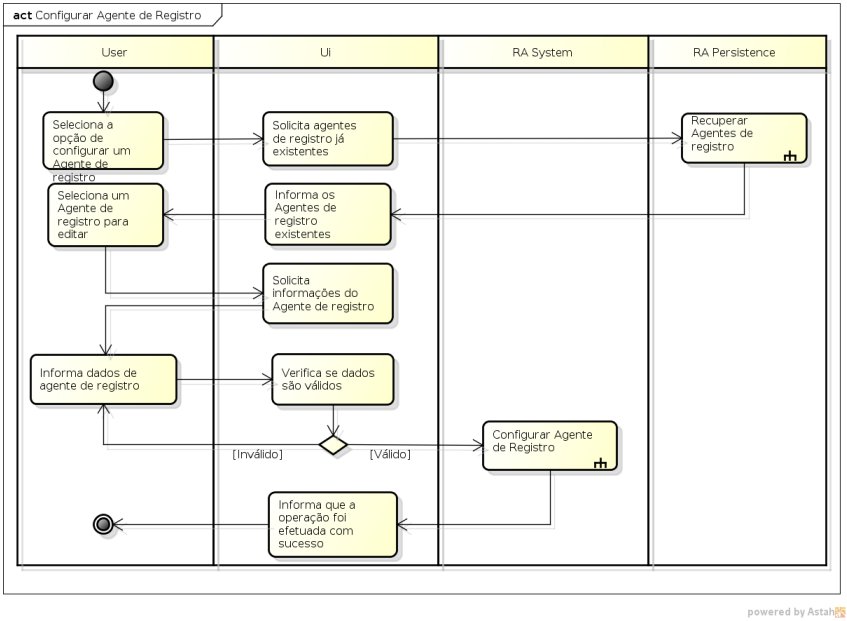


Figura 72 – “Figura referente a configuração de um Agente de Registro.”

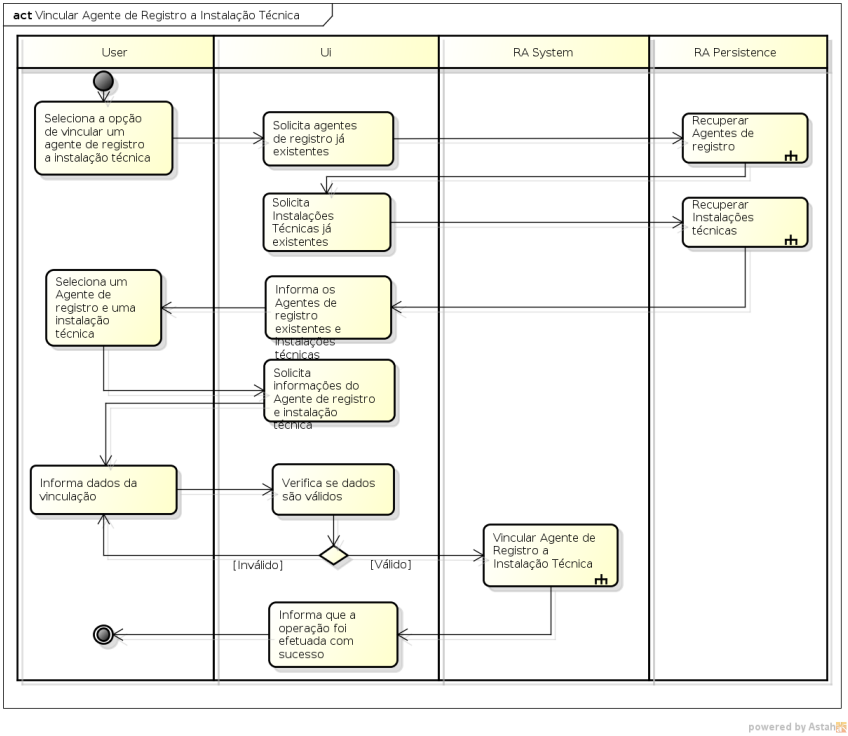


Figura 73 – “Figura referente a vinculação de um Agente de Registro com uma Instalação Técnica.”

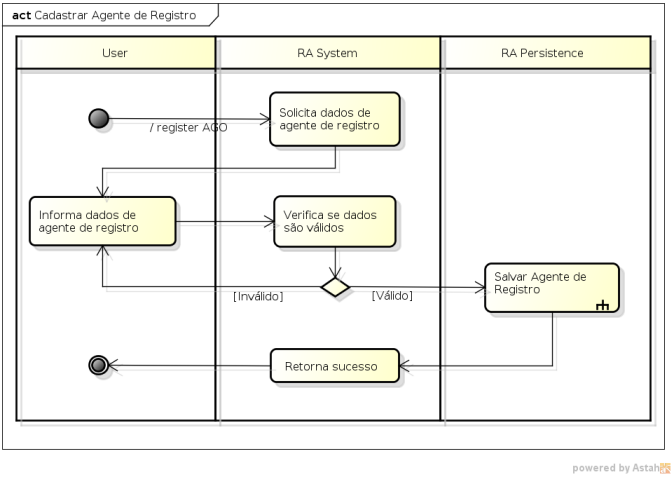


Figura 74 – “Figura referente ao cadastro de um Agente de Registro na visão do sitema.”

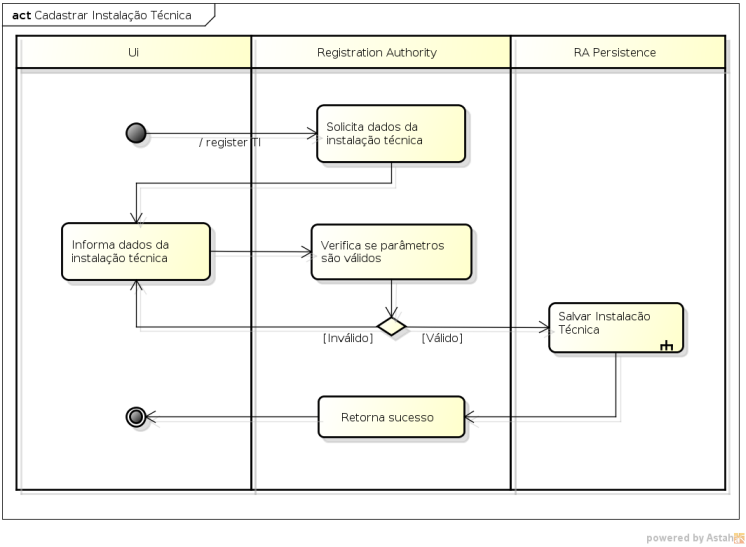


Figura 75 – “Figura referente ao cadastro de uma Instalação Técnica na visão do sitema.”

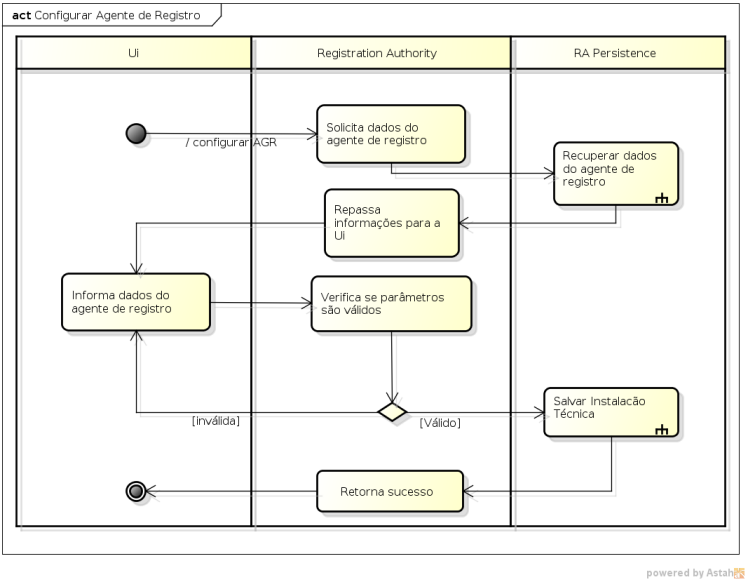


Figura 76 – “Figura referente a configuração de um Agente de Registro na visão do sitema.”

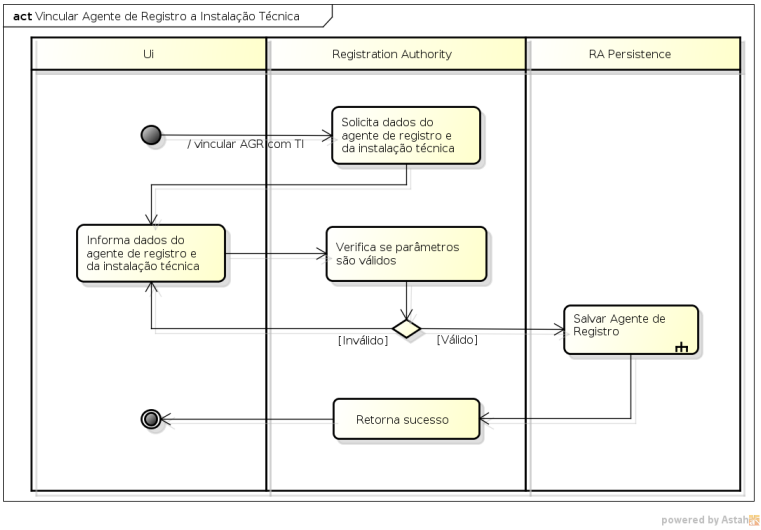


Figura 77 – “Figura referente a vinculação de um Agente de Registro com uma Instalação Técnica na visão do sitema.”

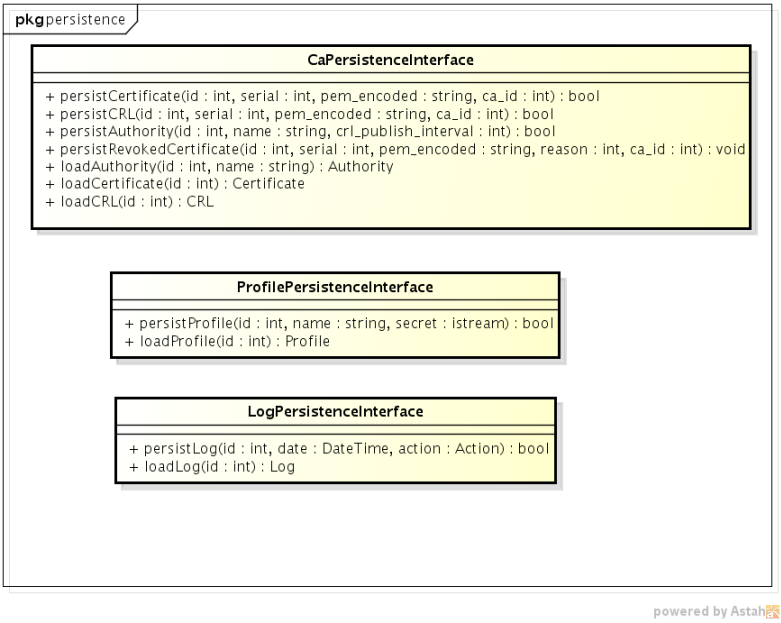
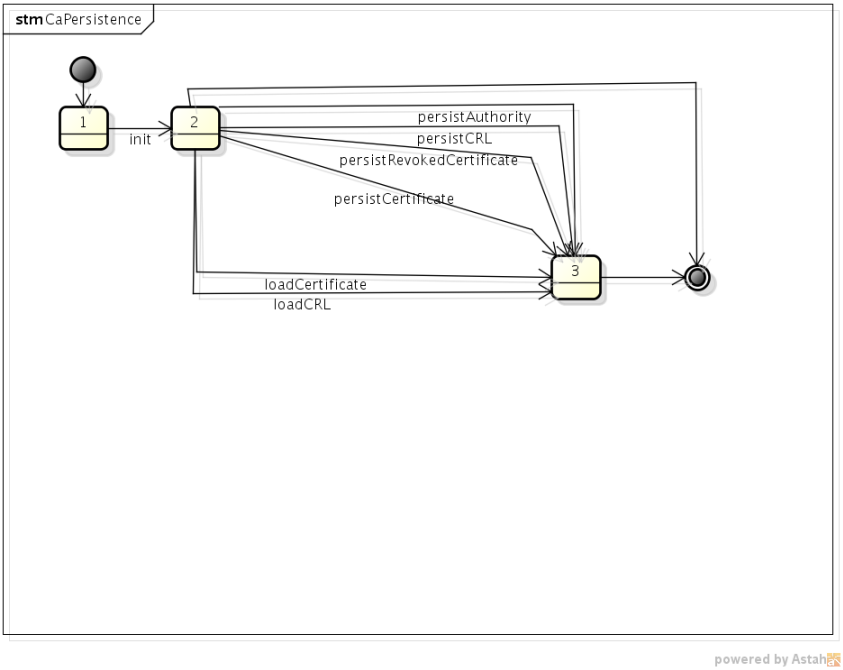


Figura 78 – “Figura referente a interface dos serviços da persistência da AC.”

Figura 79 – “Figura referente a máquina de estados dos serviços da persistência da AC.”



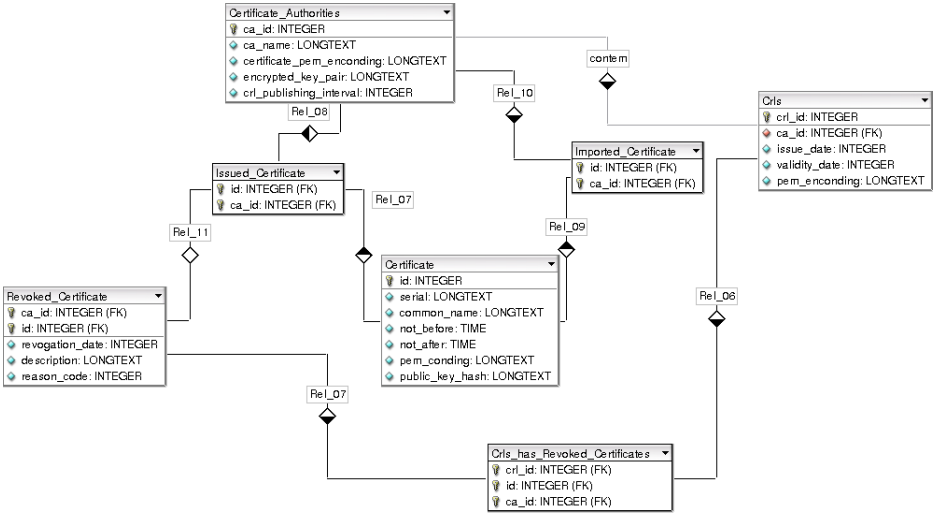


Figura 80 – “Figura referente ao modelo simples de banco de dados.”

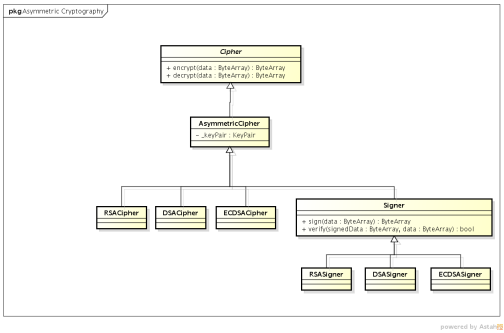


Figura 81 – “Diagrama de classe do componente Model, classes de cifradores assimétricos.”

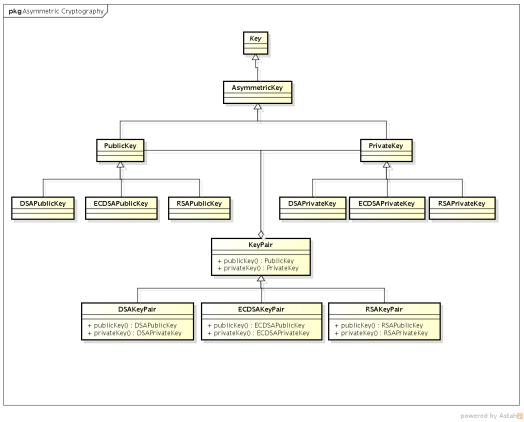


Figura 82 – “Diagrama de classe do componente Model, classes referente a chaves de algoritmos assimetricos.”

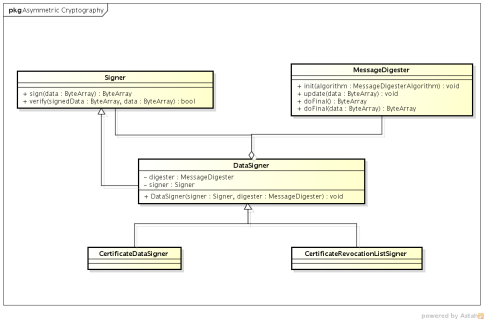


Figura 83 – “Diagrama de classe do componente Model, classes referente a assinatura em criptografia assimétrica.”

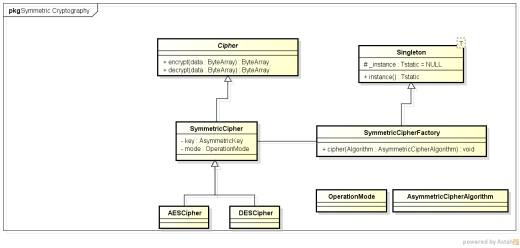


Figura 84 – “Diagrama de classe do componente Model, classe referente a cifradores simétricos”

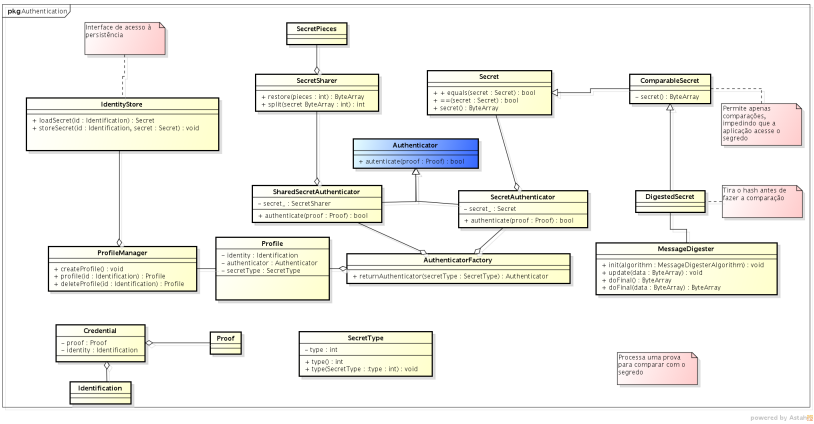


Figura 85 – “Diagrama de classe do componente Model, classes de autenticação.”

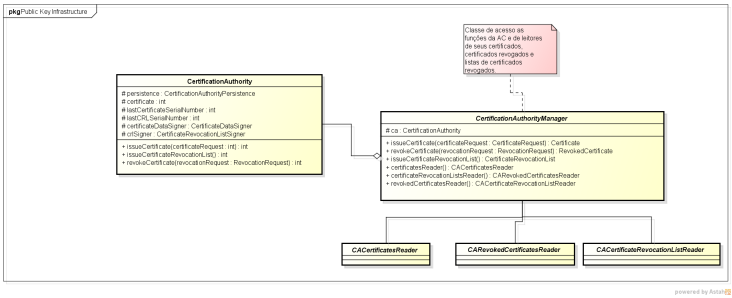


Figura 86 – “Diagrama de classe do componente Model, classes referente a funções da Autoridade Certificadora.”

REFERÊNCIAS BIBLIOGRÁFICAS

BECK, K. *Test-driven development: by example*. [S.l.]: Addison-Wesley, 2003. (The Addison-Wesley signature series). ISBN 9780321146533.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML: guia do usuário*. [S.l.]: Elsevier, 2006. ISBN 9788535217841.

CALLAS. *OpenPGP Message Format*. 2007. Disponível em: <http://www.ietf.org/rfc/rfc5280.txt>. Acesso em: 01/02/2011.

CI, J. *Jenkins*. 2011. Disponível em: <http://jenkins-ci.org/>. Acesso em: 01/10/2011.

DUBUISSON, O. *ASN.1: communication between heterogeneous systems*. [S.l.]: Morgan Kaufmann, 2001. ISBN 9780126333619.

FAYAD, M.; SCHMIDT, D.; JOHNSON, R. *Building application frameworks: object-oriented foundations of framework design*. [S.l.]: Wiley, 1999. (Wiley Computer Publishing). ISBN 9780471248750.

FEATHERS, M.; LEPILLEUR, B. *CPPunit Cookbook*. 2002. Disponível em: <http://cppunit.sourceforge.net/doc/lastest/cppunit$_c$*ookbook.html¿.Acessoem* : 20/09/2011.

GAMMA, E. *Design Patterns: Elements of Reusable Object-Oriented Software*. [S.l.]: Addison-Wesley, 1995. (Addison-Wesley Professional Computing Series). ISBN 9780201633610.

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

ICP-BRASIL, C. G. D. *GLOSSÁRIO ICP-BR - INFRA-ESTRUTURA DE CHAVES PÚBLICAS BRASILEIRAS*. Versão 1.2. Brasília, 2007. 38p.

ICP-BRASIL, C. G. D. *Manual de Condutas Técnicas 4 - Vol.1 (MCT 4 Vol.1) - Requisitos Materiais Documentos para Softwares de Assinatura*. versão 2.0. São Paulo, 2007.

ICP-BRASIL, C. G. D. *Manual de Condutas Técnicas 11 - Vol.1 (MCT 11 Vol.1) - Procedimentos de Ensaios para Avaliação de Conformidade aos Requisitos Técnicos de Softwares de AC e AR no âmbito da ICP-Brasil*. versão 1.0. [S.l.], 2010.

IETF. *RFC2511*. 1999. Disponível em: <http://tools.ietf.org/html/rfc2511>. Acesso em: 20/09/2011.

LUCIANO, D.; PRICHETT, G. Cryptology: From ceasar ciphers to public-key cryptosystems. *The College Mathematics Journal*, 1987.

MARTIN, R. *Clean code: a handbook of agile software craftsmanship*. [S.l.]: Prentice Hall, 2009. (Robert C. Martin series). ISBN 9780132350884.

OPENSSL. *OpenSSL*. 2002. Disponível em: <http://www.openssl.org>. Acesso em: 20/09/2011.

POLK, R. H. T. *Planning for PKI*. New York, USA: Ed. Wiley, 2001. 318 p.

PRESSMAN, R. *Engenharia de software*. [S.l.]: MCGRAW HILL - ARTMED, 2006. ISBN 9788586804571.

SCHNEIER, B. *Applied Cryptography*. [S.l.]: John Wiley Sons, 1996. 784 p.

SILVA, R. P. *Suporte ao desenvolvimento e uso de frameworks e componentes*. Tese (Doutorado) — Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, Brazil, 2000.

SILVA, R. P. e. *Como Modelar com UML 2*. [S.l.]: Visual Books, 2009.

SINGH, S. *O Livro dos Códigos*. São Paulo, Brazil: Ed. Record, 2001. 512 p.

SUMMERFIELD, M. *Advanced Qt Programming: Creating Great Software with C++ and Qt 4*. [S.l.]: Addison Wesley Professional, 2010. (Prentice Hall Open Source Software Development Series). ISBN 9780321635907.

TAKEUCHI, H.; NONAKA, I. *New Product Development Game*. [S.l.]: Harvard Business School Reprint, 1986. ISBN 9780000861160.

THELIN, J. *Foundations of Qt development*. [S.l.]: Apress, 2007. (Expert's voice in open source). ISBN 9781590598313.

UCHÔA, M. A. *Framework para Integração de Sistemas de Banco de Dados Heterogêneos*. Tese (Doutorado) — PUC-Rio, Rio de Janeiro, RJ, Brazil, 1999.

WANG, Y.; HU, M. Timing evaluation of the known cryptographic algorithms. In: *Computational Intelligence and Security, 2009. CIS '09. International Conference on*. [S.l.: s.n.], 2009. v. 2, p. 233 –237.