FELIPE CARLOS WERLANG, LUCAS GONÇALVES MARTINS

SISTEMA GERENCIADOR DE CERTIFICADOS OFFLINE

Florianópolis

2010

FELIPE CARLOS WERLANG, LUCAS GONÇALVES MARTINS

SISTEMA GERENCIADOR DE CERTIFICADOS OFFLINE

Melhorias em um Software de Gestão de Ciclo de Vida de Certificados, para Autoridades Certificadoras Offline

Orientador: Cristian Thiago Moecke

Co-orientador: Dr. Ricardo Felipe Custódio

UNIVERSIDADE FEDERAL DE SANTA CATARINA

Florianópolis

2010

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

Título: Sistema Gerenciador de Certificados Offline

Autores: Felipe Carlos Werlang, Lucas Gonçalves Martins

Orientador:

Cristian Thiago Moecke (UFSC)

Coorientador e Professor Responsável:

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

Banca Avaliadora:

Msc. Marcelo Carlomagno Carlos

Jonathan Gehard Kohler

DEDICATÓRIA

Dedico este trabalho à minha Família,
e principalmente a meus pais, que me
proporcionaram a educação e os valores
que são o ponto de partida de onde
trilho meus próprios caminhos.
Dedico também a meus amigos, principalmente
ao pessoal do LabSEC, que sempre estiveram
prontos para oferecer o seu apoio e
compartilhar seus conhecimentos.
Felipe Carlos Werlang

Dedico este trabalho à minha Família,
que mesmo distante
nunca deixou de estar ao meu lado.
Principalmente, aos meus pais,
que me ensinaram desde de pequeno,
a ver o mundo com os olhos de um sonhador.
Lucas Gonçalves Martins

AGRADECIMENTO

Agradecemos ao Professor Ricardo Felipe Custódio pelo apoio e incentivo a realização deste trabalho no laboratório onde é supervisor, o LabSEC. Estendemos este agradecimento aos colegas do LabSEC pelo aprendizado adquirido durante a participação no projeto.

Agradecemos ainda ao Instituto de Tecnologia da Informação (ITI) pelo apoio ao projeto João de Barro.

SUMÁRIO

LISTA DE FIGURAS

RESUMO

O Projeto João de Barro tem como objetivo a criação de uma plataforma criptográfica, formada por software e hardware, para a Autoridade Certificadora Raiz Brasileira. O software de gestão de ciclo de vida de certificados digitais da AC Raiz Brasileira foi desenvolvido pelo Laboratório de Segurança em Computação (LabSEC) da Universidade Federal de Santa Catarina (UFSC). Os autores, participantes da equipe de desenvolvimento do projeto na fase final de manutenção e implantação das versões em uso do software, apresentam melhorias implantadas neste software no período. Inclui-se aí a ramificação do software para um sistema de gestão de Autoridades Certificadoras Intermediárias, a compatibilidade do software com novos algoritmos (SHA-2 e ECDSA), mecanismos de controle de updates e outras implementações. Fruto do trabalho também é a atualização da documentação do desenvolvedor, na forma de elaboração de artefatos de engenharia de software.

Palavras-chave: Criptografia, ICP, ICP-Brasil, Certificado Digital, x509, SGC, Autoridade Certificadora, João de Barro.

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

A Medida Provisória 2.200-2, de agosto de 2001, instituiu a ICP-Brasil, a Infraestrutura de Chaves Públicas brasileira, objetivando alavancar o uso de certificação digital no governo e sociedade, como forma de agregar segurança e confiabilidade as mais diversas transações em meio digital [1].

Para operar essa infraestrutura, na época, dependia-se de soluções de software e hardware estrangeiras, de código e desenho fechados. Lançou-se então o projeto João-de-Barro, para o desenvolvimento de toda a estrutura necessária para a operação de uma ICP, porém em Software aberto, auditável e de tecnologia nacional [2].

Neste contexto, a UFSC, através do LabSEC (Laboratório de Segurança em Computação), ficou responsável pelo desenvolvimento do Software de Gestão de Ciclo de Vida de Certificados Digitais (SGC) para a AC Raiz Brasileira da ICP-Brasil. As funcionalidades e requisitos que deveriam estar presentes foram definidos pelo ITI, através de outras entidades participantes. O software recebeu o nome Ywapa ("Raiz", em Tupi-Guarani).

O presente trabalho diz respeito ao processo de finalização e entrega deste software, através de atividades desenvolvidas dentro do LabSEC. Os autores participaram ativamente do ciclo de desenvolvimento do software, implementando melhorias e correções necessárias.

1.2 OBJETIVOS

1.2.1 *GERAL*

Propor e implementar melhorias e correções no software e artefatos de engenharia de software do projeto SGC/João de Barro

1.2.2 *ESPECÍFICOS*

- Descrever o escopo de aplicação do software SGC;
- Descrever o processo de desenvolvimento do software SGC;
- Elaborar documentação do desenvolvedor atualizada para o software SGC (artefatos de engenharia de software);
- Propor e implementar o processo de ramificação do software Ywapa para o software Ywyra, que é o Software de Gestão de Ciclo de Vida de Certificados para Autoridades Certificadoras Intermediárias;
- Adicionar suporte a novos algoritmos criptográficos, em especial SHA-2 e ECDSA;
- Propor, implementar e descrever melhorias e correções no software;

1.3 JUSTIFICATIVA E MOTIVAÇÃO

A instituição da ICP-Brasil trouxe junto a inúmeros benefícios uma dependência indesejável de tecnologia estrangeira para o gerenciamento de ACs, o que não é muito interessante quando o tema envolve soberania e segurança nacional. Isso motivou a criação do Projeto João de Barro, e por consequência o SGC para a AC Raiz, mas para que o software possa atender às demandas ele precisa alcançar um estágio de maturidade, ou seja, precisa estar em uma versão estável, que não gere falhas capazes de interromper a operação, as quais seriam inadmissíveis em uma AC Raiz. Daí surge a necessidade de realizar-se a manutenção do software após a fase inicial de implementação, corrigindo as falhas encontradas e fazendo os testes necessários para garantir que nada tenha passado despercebido.

Uma vez que se tenha um sistema confiável para a operação da AC Raiz é preciso prover uma solução semelhante para as ACs intermediárias subordinadas a ela. Esta demanda motivou a ramificação do projeto do SGC para a criação de um novo software destinado a ACs intermediárias baseado no sistema original, já que as duas soluções apresentam características semelhantes, divergindo apenas no foco de operação.

Além disto, ao passar do tempo, algoritmos criptográficos eventualmente deixam de ser seguros, o que obriga o SGC a suportar algoritmos alternativos, preferencialmente os mais novos e eficientes.

Ao final do processo de manutenção e melhorias no sistema, espera-se obter uma base sólida de conhecimento e experiência nas soluções utilizadas em uma ICP, ao mesmo tempo em

que o país torna-se independente em uma área tecnológica de vital importância ao seu desenvolvimento.

1.4 METODOLOGIA

Para realizar este trabalho, foi necessário em primeiro lugar a ambientação dos autores ao software em questão e ao projeto de qual ele faz parte como um todo, afim de conhecer suas funcionalidades e aplicações.

Começou-se pela realização de testes de operação no software de forma a percorrer todas as suas funcionalidades, onde foram sendo reportadas todas as falhas encontradas. Isto ajudou para que tivesse-se pelo menos uma ideia inicial do funcionamento interno do sistema e de como se dá o processo de manutenção e melhoria deste. Em paralelo aos testes fez-se o estudo dos conceitos de segurança e certificação digital envolvidos no sistema, conhecimento este indispensável antes da realização de alguma modificação.

Em seguida foi feita a implementação de uma aplicação para a exibição do conteúdo de certificados digitais utilizando as mesmas tecnologias empregadas no SGC, afim de promover-se a familiarização com a linguagem de programação e as bibliotecas envolvidas.

Uma vez completada a fase de familiarização com o sistema e as tecnologias empregadas deu-se inicio à fase de manutenção, onde foram corrigidas as principais falhas de acordo com as demandas da gerencia do projeto e assim tornar o software estável o suficiente para entrar em operação na AC Raiz Brasileira.

Na sequencia foram analisadas em conjunto com os gerentes do projeto as alternativas mais viáveis para a implementação das melhorias do software solicitadas pelo ITI, sempre com a preocupação de minimizar o impacto das alterações e garantir assim a consistência da estrutura original.

Por fim fez-se a atualização da documentação de desenvolvimento do projeto, a fim de adequá-la à realidade do software depois das inúmeras modificações sofridas.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 CRIPTOGRAFIA

O termo Criptografia tem sua origem na junção das palavras Cripto (Escondido) e Grafia (Escrita), que vem a significar escrita escondida. Ela vem sendo usada a milênios para garantir que mensagens confidenciais transmitidas através de um meio inseguro sejam ilegíveis a alguém que as intercepte, de forma que somente o destinatário tenha acesso a seu conteúdo. A operação realizada pelo emissor de uma mensagem sobre a mesma a fim de torná-la ilegível é chamada de cifragem. A operação realizada pelo destinatário para torná-la legível novamente é chamada de decifragem [3].

Nas técnicas mais primitivas de criptografia a própria técnica precisava ser mantida em segredo. Por exemplo, se em uma mensagem cada letra fosse substituída pela letra situada 3 posições à direita no alfabeto, bastaria ao destinatário saber que ele deveria substituir cada letra pela letra situada 3 posições à esquerda para reverter o processo e obter a mensagem original. Este é um exemplo clássico conhecido como cifrador de César. Nos cifradores modernos foi introduzido o conceito de chave, a qual é usada em conjunto com os mecanismos de cifragem e decifragem, de forma que para abrir uma mensagem, um interceptador precisa não só conhecer o mecanismo, mas também ter a posse da chave [3]. O conceito pode ser demonstrado de maneira simplificada da seguinte forma:

Supondo, por exemplo, a mensagem original m = 16 e a chave k = 4, a mensagem cifrada c pode ser obtida por meio da função matemática m.k = c . Portanto c = 64.

Para recuperar a mensagem original basta realizar a operação inversa, ou seja c/k = m , onde m = 16.

Uma regra importante a ser observada é o Princípio de Kerckhoff, que diz que a segurança de um esquema de criptografia deve depender apenas do sigilo da chave, e não do sigilo dos algoritmos utilizados. Sugere-se inclusive que os algoritmos sejam publicados, possibilitando que falhas que os tornariam suscetíveis a ataques sejam identificadas mais facilmente [4].

De forma geral, a criptografia simétrica, assim chamada pois emissor e receptor fazem uso de chaves idênticas, tem como principais aplicações o envio seguro de informações sigilosas, como email e documentos, e o armazenamento seguro de dados, o que pode ser considerado como um envio no tempo onde o emissor e o receptor são a mesma pessoa [4].

Um dos grandes problemas que se enfrenta no uso de criptografia simétrica para a troca de mensagens seguras é o fato de que para duas pessoas se comunicarem dessa forma, é necessário primeiro que elas definam qual a chave a ser utilizada. Para isso elas precisam se encontrar pessoalmente ou usar algum canal seguro alternativo, o qual provavelmente é muito caro ou não está disponível já que em caso contrário não seria necessário usar a criptografia em primeiro lugar.

São inúmeros os algoritmos de criptografia simétrica existentes atualmente [5], mas merecem destaque o Data Encryption Standard (DES) [6] e seu sucessor, o Advanced Encryption Standard (AES) [7].

2.1.1 *CRIPTOGRAFIA ASSIMÉTRICA*

Em 1976, os pesquisadores norte-americanos Whitfield Diffie e Martin Hellman publicaram um artigo chamado *"New Directions in Cryptography"* onde descreveram uma alternativa para resolver o problema da troca de chaves entre emissor e receptor, o qual acabou dando origem um novo tipo de criptografia, conhecida como criptografia assimétrica. Nesse novo modelo ao invés de uma única chave é usado um par de chaves, onde uma delas é publica(pode e deve ser divulgada) e a outra privada(secreta). Se geradas sob determinadas propriedades matemáticas, essas chaves possibilitam que o que for cifrado com a chave pública só possa ser decifrado com a chave privada, e vice-versa [8].

Assim como a criptografia convencional, a criptografia assimétrica também tem como funcionalidade inicial prover sigilo na troca de informações, mas ela também possui propriedades adicionais muito interessantes que promoveram o aumento do leque de aplicações deste tipo de criptografia. Essas propriedades são a autenticidade e o não-repúdio [5].

Podemos ilustrar o conceito e as propriedades da criptografia assimétrica com as seguintes situações:

- **Sigilo**

  Alice deseja enviar mensagem sigilosa a Beto. Para isto basta que Alice cifre a mensagem com a chave pública de Beto. Apenas Beto, que é o custodiante da chave privada poderá

decifrá-la.

- **Autenticidade**

  Beto deseja determinar a autenticidade da mensagem de Alice. Se Beto conseguir decifrar a mensagem com a chave pública de Alice quer dizer que a mensagem realmente veio dela, pois Alice é a única custodiante da chave privada utilizada para cifrar a mensagem.

- **Não-Repúdio**

  Alice não pode repudiar a autoria de mensagem enviada a Beto. De forma semelhante à propriedade de autenticidade, uma vez que Beto conseguir decifrar a mensagem com a chave pública de Alice, esta não poderá negar a autoria da mensagem, já que por ser a única custodiante da chave privada é também a única capaz de ter gerado a mensagem.

Apesar de possuir um número maior de aplicações e ser mais flexível em relação a troca de chaves, a criptografia assimétrica ainda apresenta uma desvantagem em relação à simétrica. Operações criptográficas assimétricas tem um custo computacional muito mais elevado do que operações simétricas. Devido a essa desvantagem muitas aplicações costumam usar os dois tipos de criptografia em conjunto. Por exemplo, criptografia assimétrica pode ser usada para o estabelecimento de uma chave de sessão (simétrica) entre uma aplicação cliente e um servidor, permitindo que eles possam comunicar-se de forma segura [4]. Numa descrição simplificada, o servidor disponibiliza sua chave pública ao cliente, que então define uma chave simétrica, a qual cifra com a chave pública do servidor para em seguida enviá-la ao mesmo. O servidor decifra a chave de sessão com sua chave privada e a partir desse momento cliente e servidor podem proteger sua comunicação por meio de criptografia simétrica [5].

Outro aspecto importante a ser lembrado é que para que as propriedades da criptografia assimétrica possam ser utilizadas 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, a qual descrita na seção 2.3.

2.2 ASSINATURA DIGITAL

Rivest, Shamir e Adleman propuseram em 1977 o primeiro método de implementação de um sistema criptográfico baseado em chaves públicas, conhecido por RSA. Sua segurança tem base na dificuldade da fatoração de números grandes, e funciona de forma análoga ao algoritmo

para troca de chaves, proposto por Diffie e Hellman [9], com a diferença de implementar outras funcionalidades, como a assinatura digital.

Como descrito na seção 2.1.1, a criptografia assimétrica provê a funcionalidade de autenticação, através da cifragem de um mensagem com a chave privada de quem deseja ser autenticado. Ou seja, cifrar um documento eletrônico com uma chave privada, funciona como uma assinatura que identifica uma pessoa em um documento de papel.

É importante destacar também que para uma assinatura ter realmente alguma utilidade é preciso garantir a integridade do documento, ou seja, é preciso garantir que seja possível detectar facilmente qualquer alteração que o documento sofra posteriormente à assinatura. Em documentos eletrônicos isto pode ser feito por meio de uma função de hash.

As próximas subseções explicam o funcionamento de uma função de hash, e como foi padronizada a assinatura digital.

2.2.1 *HASH*

A função de hash, ou de resumo criptográfico, quando aplicada em um documento digital, retorna uma sequência de bits de tamanho fixo, geralmente menor que o tamanho do documento. Além disso ela é uma função de ”mão única”, ou seja, não existe uma função inversa que receba como entrada o hash e retorne o documento original [5].

Uma vez que se calcula o hash de um documento, se o mesmo sofrer qualquer modificação, por menor que seja, um novo cálculo irá gerar um resultado totalmente diferente do anterior. Essa característica é que faz dele uma ótima ferramenta para a garantia de integridade de documentos.

Quatro propriedades são necessárias para que a função de hash possa ser considerada segura:

- Deve ser difícil recuperar o documento original a partir do hash.
- Qualquer alteração no documento deve alterar completamente o hash resultante. Conhecido como efeito avalanche.
- Deve ser difícil de se obter duas mensagens com o mesmo hash.
- O cálculo do hash deve ser simples e rápido de ser feito.

Onde entende-se por difícil, procedimento considerado de grande custo computacional, ou seja, não possui solução em tempo polinomial.

Atualmente, as implementações de algoritmos para o cálculo de resumos criptográficos mais usadas são o MD5 e o SHA-1. Porém, o NIST (National Institute of Standards and Technology), órgão responsável pela padronização na área de Tecnologia dos Estados Unidos da América, publicou um documento de recomendações para a troca de algoritmos criptográficos [10], onde é recomendado o uso do SHA-2 e o abandono completo do SHA-1 a partir de 2011.

2.2.2 *PADRONIZAÇÃO DA ASSINATURA DIGITAL*

Como descrito na seção 2.1.1, as operações criptográficas assimétricas possuem um custo computacional muito alto. Por esse motivo, a realização da assinatura (cifragem com a chave privada) de um documento muito grande, poderia levar muito tempo. Para diminuir esse tempo e ao mesmo tempo garantir a integridade dos documentos, passou-se a fazer uso de funções de resumo criptográfico no processo de assinatura digital. Em vez de assinar o documento em si, passou-se a fazer a assinatura de um hash, que é considerada equivalente a assinatura do documento correspondente.

Para garantir a interoperabilidade entre sistemas e identificar os algoritmos de assinatura digital realmente seguros, o NIST publicou em 1994 o documento *Digital Signature Standard* [11], com a padronização do uso da assinatura digital para os algoritmos RSA e DSA. Esse documento já recebeu três revisões, onde foram adicionadas novas padronizações e incluído um novo algoritmo, o ECDSA, baseado em curva elípticas [12].

A assinatura digital de um documento eletrônico, e a validação dessa assinatura, ficaram da seguinte forma:

Assinatura: Fazendo uso de uma função de hash segura o suficiente [10], deve-se calcular o resumo criptográfico do documento que se deseja assinar, e então aplicar a esse resumo o algoritmo de assinatura digital, fazendo uso da chave privada do assinante.

Validação: Para fazer a validação da assinatura digital são necessárias quatro informações: O documento que foi assinado; o hash assinado; a função de hash usada para calcular o resumo criptográfico e por fim a chave pública do assinante. Com esses parâmetros, procede-se da seguinte forma: Usa-se a chave pública para decifrar o hash assinado, em seguida aplica-se a mesma função de hash no documento e então compara-se o hash decifrado com o hash calculado. Se forem iguais a assinatura é valida, caso contrário algum dos parâmetros não está correto.

2.3 INTRAESTRUTURA DE CHAVES PÚBLICAS

Apesar de oferecer a possibilidade de cópia e distribuição, o conceito de chave pública não dá, por si só, o suporte necessário à identificação do real dono da chave privada a ela correspondente. Em outras palavras, alguém mal intencionado poderia distribuir uma chave pública em nome de outra pessoa, e dessa forma conseguir acesso a informações confidenciais à ela destinadas.

Uma solução para este problema é a utilização de uma Infraestrutura de Chaves Públicas, ou ICP, que consiste em um conjunto de componentes que interagem entre si para identificar, de forma confiável e segura, o custodiante de um chave privada. Normalmente isto se dá através de uma terceira parte confiável [13].

Dentre os vários formatos de ICP existentes no mercado, este trabalho foca-se na abordagem do formato x509 [14]. Nos próximos tópicos serão descritos os componentes de uma ICP e como eles se relacionam.

2.3.1 *CERTIFICADO DIGITAL*

O certificado digital é o elemento básico de uma ICP. Trata-se de um documento totalmente digital que contém uma chave pública e informações sobre o custodiante da chave privada correspondente, ou seja, ele relaciona um par de chaves com o seu dono [13].

Os recursos da ICP trabalham para assegurar a autenticidade e a validade do certificado digital durante todo seu ciclo de vida. Para que isto seja possível, certos campos precisam ser adicionados ao certificado. Apesar de estarem todos descritos na RFC-5280 [14], dentre os principais pode-se citar: o número de série, que identifica unicamente o certificado; o campo de assunto, que mantém as informações do dono do certificado; o campo de emissor, com as informações do responsável pela autenticidade dos dados no certificado; dois campos contendo a data inicial e a data final da validade do certificado; e a assinatura digital, feita pelo emissor, que dá autenticidade ao certificado e o protege de alterações.

Somente o certificado digital já resolve muitos dos problemas relativos a segurança no uso de chaves assimétricas, mas alguns deles persistem, como por exemplo a possibilidade de um par de chaves vir a ficar comprometido ou simplesmente deixar de ser usado. Logo, certificados digitais relacionados a pares de chaves nestas condições precisam ser identificados, de forma a não serem mais considerados válidos. A subseção 2.3.2 descreve como isso é feito, através dos conceitos de revogação de certificados e de Lista de Certificados Revogados.

2.3.2 *LISTA DE CERTIFICADOS REVOGADOS*

A lista de certificados revogados, ou LCR, como o próprio nome diz, é uma lista com os números seriais dos certificados que não são mais válidos e que ainda não estão expirados. É da responsabilidade do emissor do certificado gerar essa lista e divulgá-la constantemente, normalmente pela internet, e é da responsabilidade do usuário do certificado verificar se o mesmo não está presente na última LCR gerada pelo emissor [13].

Os motivos que justificam a inserção de um certificado na LCR dependem das políticas praticadas pela ICP. Por exemplo, um certificado emitido por uma empresa para um de seus funcionários pode ser inserido na LCR pela própria empresa caso o mesmo venha a ser demitido, mas é possível também que o dono do certificado venha a pedir para empresa revogar seu certificado por motivos diversos, como o comprometimento da chave privada.

Quais campos podem ser incluídos na LCR, além do modo como estes devem ser apresentados, está descrito na RFC-5280 [14], mas dentre os principais pode-se citar: o campo de emissor, com as informações do responsável pela autenticidade dos dados na LCR; dois campos de data, contendo a data de emissão da LCR e a data da próxima emissão; a lista dos seriais dos certificados revogados; e a assinatura digital, feita pelo emissor, que dá autenticidade a LCR e a protege de alterações.

Apesar de ser um conceito simples, a LCR pode ser considerada um ponto fraco da ICP, já que em muitos casos, é preciso estar conectado a internet para se obter a última LCR emitida. Além disso, existe a necessidade de se emitir uma nova LCR sempre que a atual estiver para atingir seu prazo de validade, e esse controle de continuidade das LCRs pode não ser simples, podendo causar a existência de mais de uma LCR válida ao mesmo tempo, o que por sua vez possibilita que um certificado seja válido e inválido ao mesmo tempo.

2.3.3 *AUTORIDADE CERTIFICADORA*

Como mencionado nas seções anteriores, existem vários formatos de ICP. Um exemplo deles é o PGP [15], que possui uma infraestrutura em rede onde não há níveis hierárquicos e a validade do certificado se dá na quantia de pessoas que confiam, ou seja, assinam aquele certificado. Esse formato se baseia em uma rede de confiança, parecida com a rede de confiança entre pessoas, mas deixa algumas brechas que dificultam a validação do certificado e o impedem de ter validade legal.

O advento da Autoridade Certificadora, ou AC, se deu para centralizar o ponto de confiança dos certificados, facilitando sua validação e lhe garantindo um certo valor legal. Uma AC é o

conjunto de hardware, software e pessoas que a operam. Ela é identificada pelo seu nome e sua chave pública [13]. As três principais funções da AC dentro da ICP são:

- Emissão de Certificado: Criar e assinar certificados digitais.
- Revogação de Certificado: Inserir o serial de um certificado na Lista de Certificados Revogados.
- Emissão de LCR: Assinar e publicar a LCR.

Para não haver uma sobrecarga de emissões de certificados e LCRs em um único ponto, cria-se uma hierarquia de ACs, onde no topo se encontra a AC-Raiz, autoassinada, e logo um nível abaixo encontram-se as Autoridades Certificadoras Intermediárias, com certificados assinados pela AC Raiz. Ao final dessa árvore hierárquica encontram-se as ACs Finais, responsáveis pela emissão de certificados para usuários finais.

2.3.4 *AUTORIDADES REGISTRADORAS*

A Autoridade Registradora, ou AR, é responsável pela interface entre o requerente de certificado e a Autoridade Certificadora [16]. Ou seja, a AR faz o serviço de verificação dos dados de uma entidade que deseja ter um certificado de alguma das ACs para qual a AR provê o serviço.

As ARs otimizam os serviços das ACs, deixando-as livres para realizar apenas as suas funcionalidades básicas. Além disso, aumentam a área geográfica de cobertura da AC, já que não será preciso que o requerente vá até o prédio onde se localiza a Autoridade Certificadora, e sim a qualquer instalação de AR de confiança dessa AC.

Uma Autoridade Certificadora que faz uso de ARs, define uma lista de ARs de sua confiança. Uma entidade que deseja um certificado dessa AC deve ir até uma das ARs de confiança e realizar a requisição do certificado. Nesse momento serão feitas as verificações necessárias, normalmente através de documentos de identificação, para autenticar o requerente do certificado. Com os dados validados, a AR envia a requisição do certificado para AC através do Protocolo de Gerenciamento de Certificados (*Certificate Management Protocol* - CMP) para que a Autoridade Certificadora emita o certificado do requerente [17].

2.3.5 *REQUISIÇÃO DE CERTIFICADO*

Para facilitar o pedido de emissão de certificado, criou-se o conceito de requisição de certificado, um arquivo que contém informações relativas ao custodiante de um par de chaves, junto

com a chave pública.

Uma entidade que deseja um certificado de uma AC, deve primeiramente gerar um par de chaves e criar a requisição com suas informações e sua chave pública. Após esse procedimento, ela deve realizar a assinatura da requisição fazendo uso da chave privada correspondente a chave pública na requisição, ou seja, autoassiná-la. Depois disso envia-se a requisição a uma Autoridade Certificadora, diretamente ou por intermédio de uma Autoridade Registradora, para que o certificado seja emitido.

A especificação completa dos campos de uma Requisição de Certificado pode ser encontrada na RFC-2986 [18], mas a estrutura do arquivo de requisição é bem parecida com a estrutura do certificado digital, com destaque apenas à ausência de alguns campos que só fazem sentido após a emissão, como o campo de emissor.

3 SISTEMA GERENCIADOR DE CERTIFICADOS

Sistema Gerenciador de Certificados (SGC) é o software responsável pela gestão do ciclo de vida de uma ou mais Autoridades Certificadoras, abrangendo desde a criação e operação até o eventual encerramento das atividades das mesmas. De forma geral é um software que dá suporte à criação e exclusão de instâncias de ACs, emissão e revogação de certificados digitais e emissão de LCRs.

O desenvolvimento do SGC para a ICP-Brasil teve início em 2005 e seguiu as especificações formuladas pelo CASNAV (Centro de Análises de Sistemas Navais - Marinha Brasileira)[1], uma das entidades participantes do Projeto João de Barro. A plataforma escolhida para o software foi a linguagem de programação C++ e as bibliotecas de interface gráfica QT, o banco de dados PostgreSQL e o sistema operacional Red Hat Linux. Para as operações criptográficas optou-se pelo uso das bibliotecas do Projeto OpenSSL. Mas devido às dificuldades encontradas no uso direto do OpenSSL, que é escrito em C e portanto não orientado a objetos, foi também usada uma biblioteca criptográfica em C++ para encapsular as funções do OpenSSL, chamada LibCryptoSEC, a qual foi desenvolvida e publicada pelo LabSEC [2].

Em 2008 a primeira versão estável do sistema entrou em operação, sendo utilizado na criação da nova AC Raiz Brasileira. Devido ao sucesso da sua utilização surgiu o interesse do seu uso também em ACs de nível intermediário. Isso levou a uma ramificação do sistema, chamada de Ywyra, voltada para ACs Intermediárias. Desta forma a equipe passou a trabalhar em dois sistemas paralelamente, Ywapa (AC Raiz) e Ywyra (ACs Intermediárias).

3.1 FUNCIONALIDADES

As funcionalidades de um SGC podem ser distinguidas entre gerais, que dizem respeito ao sistema como um todo, ou exclusivas, que são restritas ao modo de administração ou ao modo de operação.

[1] https://www.casnav.mar.mil.br
[2] https://projetos.labsec.ufsc.br/libcryptosec

3.1.1 *GERAIS*

Suporte a MSC via Engine OpenSSL: O SGC suporta o uso de um Módulo de Segurança Criptográfico (MSC) para o armazenamento das chaves das ACs. Isto é feito por meio de uma Engine OpenSSL que é uma interface de comunicação utilizada por vários MSCs. Desta forma para utilizar um MSC diferente basta indicar nas configurações do sistema o caminho para o arquivo da respectiva engine [19]. A seção 5.3 descreve melhorias implementadas nesta funcionalidade.

Registro de Eventos: A fim de proporcionar subsídios para a realização de auditorias na operação das ACs, o sistema mantém o registros de eventos (Logs) de todas as operações relevantes, entenda-se aqui de maneira geral qualquer operação que cause alguma mudança no estado do sistema, bem como tentativas canceladas ou que falharam. Além de poderem ser exportados para arquivos, esses logs também são utilizados para a geração de relatórios gerenciais. A seção 5.4 descreve melhorias implementadas nesta funcionalidade.

Autenticação via Segredo Compartilhado: A autenticação dos membros de grupos dos perfis de administração e operação utiliza a técnica de segredo compartilhado, onde cada membro utiliza seu smart card para decifrar uma parte do segredo, que ao ser remontado libera acesso aos dados do perfil na base de dados.

Backup: Para prevenir a perda dos dados das ACs existe a possibilidade de geração de um backup completo das bases de dados do SGC, onde os dados sigilosos são cifrados e o pacote completo é assinado ao final para garantir a integridade. A restauração de backups exige a autenticação do perfil de administração. Backups podem ainda ser restaurados em versões mais recentes do software em relação aquelas em que foram gerados.

Atualização Automática de Esquema de BD: Quando uma versão mais nova do software é instalada o sistema verifica se houve alguma alteração na estrutura das bases de dados e caso necessário procede automaticamente com as modificação necessários nas bases de dados existentes para que elas atendam às necessidades do novo software. O mesmo processo ocorre na restauração de backups gerados em versões mais antigas. A seção 5.1 descreve a implementação desta funcionalidade.

3.1.2 *MODO DE ADMINISTRAÇÃO*

Gerência de Autoridades Certificadoras: Fora algumas tarefas de configuração, as principais funcionalidades presentes no modo de administração são a criação e a exclusão de Autoridades Certificadoras. Na criação de ACs existe uma diferença essencial entre o sistema

Ywapa e o sistema Ywyra, uma vez que no primeiro ao final é emitido um certificado autoassinado enquanto no segundo é emitida apenas uma requisição de certificado, a qual precisa ser exportada para que uma outra AC emita o respectivo certificado, e este por sua vez é importado no sistema finalizando o processo de criação.

3.1.3 *MODO DE OPERAÇÃO*

Emissão de Certificados: As requisições de certificados digitais são importadas a partir de arquivos e seus campos podem ainda ser modificados antes da emissão dos respectivos certificados. Depois de emitidos o certificados podem ser exportados a qualquer momento.

Revogação de Certificados: Cada AC pode revogar, se necessário, os certificados por ela emitidos que ainda não estejam com seu período de validade expirado. Isto se traduz na adição do certificado à lista de certificados revogados.

Emissão de LCRs: Para efetivar de fato a revogação dos certificados é necessário emitir a Lista de Certificados Revogador (LCR) e exportá-la. Existe ainda a possibilidade de alterar o intervalo de emissão da LCR, indicando o limite de tempo em que uma nova Lista de Certificados Revogados deve ser publicada.

Suporte a Modelos de Certificados: Como praticamente todos os certificados emitidos por uma AC tem vários campos com dados padronizados, o sistema oferece a possibilidade de se criar modelos padronizados de certificados. Desta forma na emissão é possível selecionar um modelo existente fazendo com que os dados de vários campos do certificado sejam preenchidos automaticamente.

Geração de Relatórios Gerenciais: Existe ainda a possibilidade de gerar relatórios gerenciais contendo os registros da operação de uma AC, os quais fazem uso de modelos de relatório previamente configurados pelos próprios operadores. Na criação de modelos são definidos os tipos de logs que estarão presentes nos relatórios. Os relatórios separam entre si as operações realizadas pelos perfis de administração e operação e usam diferentes colorações para marcar ações normais e críticas de acordo com as informações presentes nos logs.

3.2 PROCESSO DE DESENVOLVIMENTO

Ao entrar para o projeto, no início de 2008, os autores assumiram o cargo de programadores, sendo responsáveis pela correção de bugs e implementação de melhorias no sistema, uma vez que o estágio de implementação inicial já estava finalizado. O restante da equipe era constituída

por Gerente de Projeto, Gerente de Configuração e Gerente de Qualidade.

Nessa estrutura o Gerente de Projeto é o encarregado de fazer o intermédio entre a equipe e o ITI, coordenar as atividades dentro dos prazos estabelecidos e lançar as versões do software. O Gerente de Configuração é responsável por preparar o ambiente de desenvolvimento, tanto o hardware quanto as ferramentas de software a serem utilizadas, tendo uma atuação mais acentuada no início do projeto. Por último o Gerente de Qualidade atua na revisão das correções e melhorias implementadas pelos programadores e na aplicação de testes, a fim de garantir a confiabilidade do sistema.

O fluxo de interações entre os membros da equipe a cada nova tarefa seguiu praticamente sempre o mesmo padrão. Primeiramente o ITI passava demandas de melhorias para o gerente de projeto, que então registrava a pendência (ticket) no sistema de gerenciamento de projeto (Trac). Em seguida um dos programadores assumia a responsabilidade pelo ticket, também via Trac, e efetuava as alterações no código. Ao finalizar as mudanças o programador repassava o ticket ao gerente de qualidade, que analisava a solução adotada a fim de verificar a corretude e a coerência com a estrutura do sistema. Em caso de desaprovação o gerente de qualidade devolvia a posse do ticket ao programador. Caso contrário o ticket era fechado. Na correção de bugs também ocorria a troca de posse dos tickets entre programadores e gerente de qualidade a exemplo da implementação de melhorias. A diferença aqui é que qualquer membro da equipe podia registrar um ticket de correção. A Figura 1 ilustra essa interação.

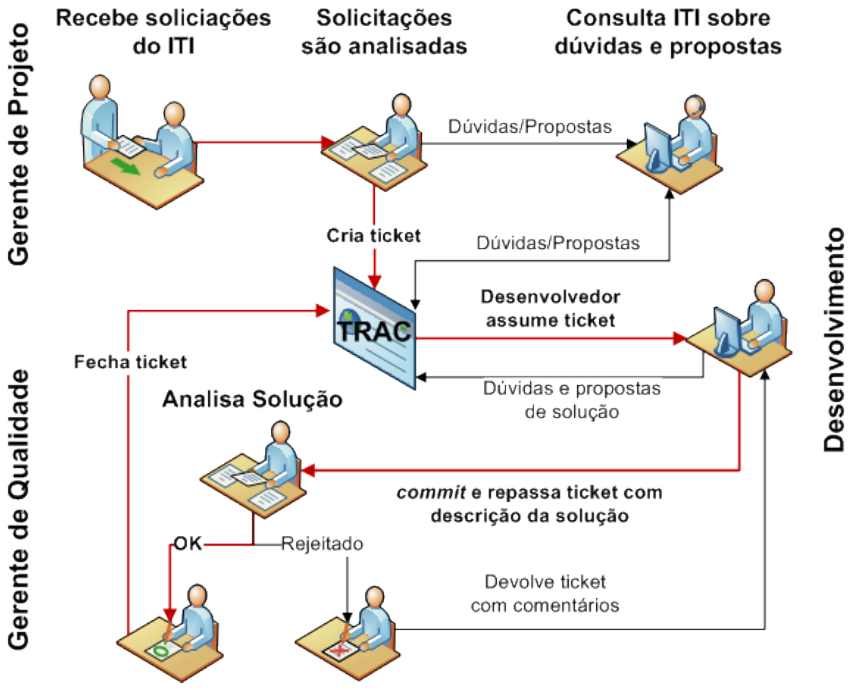

Figura 1: Fluxo de interações da equipe

De maneira geral, para cada nova versão do sistema a ser implementada, seguiram-se os seguintes passos: Primeiramente o gerente de projeto avaliava a importância de cada ticket, e após consultar a equipe sobre o tempo e esforço que o mesmo demandaria, definia se ele seria fechado nesta versão ou então postergado. Em seguida passava-se para a fase de implementação, e à medida que todos os tickets eram aprovados pelo gerente de qualidade gerava-se a nova versão. Mas para poder ser publicada, antes esta versão do software ainda passava por uma bateria de testes, da qual participava toda a equipe, e onde se fazia o uso do documento de plano de testes. Quando alguma falha grave era encontrada a correção era feita imediatamente e a versão era gerada novamente. Para erros menos graves, que não resultavam em falhas na execução ou perda de integridade dos dados, eram registrados novos tickets para versões futuras.

Ao longo do processo de desenvolvimento, foram mantida também as atividades de atualização da documentação de desenvolvedor de reformulação e incremento do plano de testes, descritas nas subseções 3.2.1 e 3.2.2.

Para manter um histórico de modificações e garantir a consistência dos arquivos de código fonte quando editados por mais de um membro da equipe ao mesmo tempo foi utilizada durante todo o processo uma ferramenta de controle de versões, o Subversion (SVN).

3.2.1 *DOCUMENTAÇÃO DE DESENVOLVEDOR*

No início do desenvolvimento do SGC foram criados artefatos de engenharia de software baseados na especificação de requisitos fornecida pelo CASNAV, os quais constituíam a documentação utilizada como guia pela equipe para a implementação do software. Dentre esses artefatos destacam-se diagramas de classes, diagramas de casos de uso e diagramas de atividade.

Durante a etapa de programação muitos itens presentes na especificação foram reavaliados, de modo que ao final de 3 anos de projeto a documentação havia ficado defasada com relação a realidade do software implementado. Para resolver esse problema o gerente do projeto procedeu com a revisão da especificação de requisitos, a qual contém todos os casos de uso do sistema, e uma vez que isto estava pronto coube aos autores fazer a atualização dos diagramas. Como os diagramas originais estavam muito defasados e a ferramenta escolhida para a geração dos novos diagramas não suportava o formato dos arquivos antigos, decidiu-se por abandoná-los e confeccionar diagramas de atividade, de classes e de bases de dados a partir do zero, baseados na nova especificação.

A ferramenta de criação de diagramas possuía a funcionalidade de engenharia reversa, pos-

sibilitando gerar os diagramas de classes à partir do código fonte do software, mas os diagramas resultantes, apesar de apresentarem todas as classes e suas ligações corretamente, ficavam com uma orientação totalmente confusa. Foi necessário reposicionar todas as classes dentro do diagrama manualmente. As classes foram agrupadas de acordo com o pacote a qual pertenciam na estrutura do software e cada grupo foi colorido unicamente, facilitando a visualização e compreensão do diagrama.

Nos diagramas de bases de dados também foi utilizada engenharia reversa, sendo estes construídos a partir das instruções de geração de tabelas extraídas do código fonte, mas foi necessário corrigir os diagramas gerados pois praticamente cada atributo de cada tabela teve que ser editado para deixá-los compatíveis com a tipagem do PostgreSQL. Além disso todas as restrições que impedem inconsistência de dados a nível de banco de dados tiveram que ser inseridas manualmente nas configurações das tabelas. As restrições não aparecem de forma visual no diagrama, mas são necessárias caso se deseje extrair as instruções DDL a partir dele, o que é uma possibilidade oferecida pela ferramenta. Essas instruções são usadas para gerar a estrutura das bases no Banco de Dados.

Os diagramas de atividade foram gerados manualmente, um para cada caso de uso do sistema. Estes diagramas receberam atenção especial, pois seguem de forma simplificada o fluxo de execução do software, característica que os torna muito úteis para alguém que precise entender o funcionamento do código fonte. Tal utilidade se torna evidente quando existem mudanças de equipe ao longo do projeto. Para garantir que se tivesse o máximo possível de fidelidade entre os diagramas e o funcionamento real do software, cada um deles, depois de ser terminado, era passado para o outro membro da dupla que então avaliava sua corretude e clareza e se necessário fazia as devidas correções. Por fim o resultado ainda passava pela avaliação do gerente de qualidade.

Além dos diagramas também foi feito um levantamento de todos os eventos de log existentes no sistema, que são separados por tipo de evento, e estes por sua vez por tipo de ação. Esta listagem é apresentada juntamente com o restante da documentação no Manual de Desenvolvedor.

3.2.2 *PLANO DE TESTES*

Para facilitar o diagnóstico de falhas no software, e também para tornar esse diagnóstico sistemático foi utilizado desde o início do projeto o documento de Plano de Testes.

Em seu estado atual o Plano de Testes segue os casos de uso do sistema, chamados aqui de

casos de teste. Ele descreve o roteiro de passos a serem seguidos em cada caso de teste para garantir que o software está se comportando da forma esperada. Junto ao roteiro existe uma tabela contendo as entradas, dados ou operações válidos e inválidos, as saídas esperadas para cada entrada, e um campo a ser preenchido com o resultado do teste. O documento é apenas uma forma de checklist, uma vez que os comportamentos inadequados e falhas encontradas ao ser percorrido são registrados no Trac do projeto, na forma de tickets.

Ao longo do projeto o documento foi reformulado pelos gerentes de projeto e qualidade a fim de torná-lo mais prático e completo, cobrindo o maior número possível de possibilidades de falha, além de ter sido incrementado várias vezes à medida que novas funcionalidades foram adicionadas ao sistema.

4 *RAMIFICAÇÃO DO SGC EM YWAPA E YWYRA*

Já se cogitava, durante o decorrer do projeto, a expansão das funcionalidades do Ywapa, para que, além de Autoridades Certificadoras Raízes, ele desse suporte também a Autoridades Certificadoras Intermediárias. Uma demanda oficial para essa expansão surgiu em 2008, quando o ITI, que já fazia uso do software na AC Raiz Brasileira, decidiu que iria usá-lo também em ACs Intermediárias no futuro.

As próximas seções descrevem como foi o processo de desenvolvimento dessa nova demanda que acabou dando origem a um novo sistema, o Ywyra, com foco especial em Autoridades Certificadoras Intermediárias. A Figura 2 ilustra as etapas da ramificação do sistema.

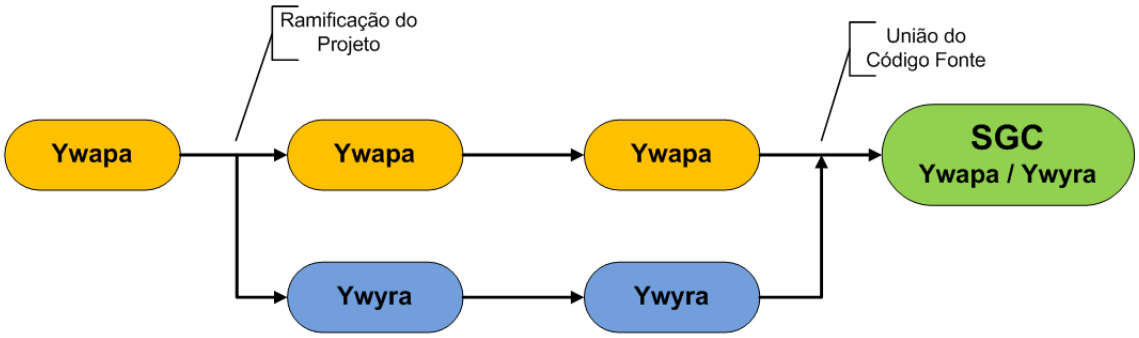

Figura 2: Ramificação do SGC em Ywapa e Ywyra

4.1 PROCESSO DE DESENVOLVIMENTO DO YWYRA

Pela natureza da Infraestrutura de Chaves Públicas, por questões de segurança, uma Autoridade Certificadora Raiz costuma ser completamente isolada em todos os meios, tornando difícil conceber uma AC Intermediária compartilhando as instalações destinadas a uma AC Raiz. Além disso, existia o interesse em que o software de AC Raiz apresentasse apenas as funcionalidades específicas, necessárias para a operação da AC Raiz e nada mais.

Por essas características, decidiu-se pela criação de um novo projeto, que se responsabi-

lizaria somente por gerenciar Autoridades Certificadoras Intermediárias, ao invés de inserir essa funcionalidade no Ywapa. O nome dado ao projeto foi Ywyra, que vem do tupi-guarani e significa árvore em português.

Uma vez tomada a decisão de desenvolver um novo projeto, criou-se uma proposta com os novos casos de uso que cobririam as funcionalidades necessárias para gerenciar Autoridades Certificadoras Intermediárias. Devido a grande semelhança entre o Ywapa e o Ywyra, a maior parte dos casos de uso, que foram criados no desenvolvimento do Ywapa, poderiam ser usados na concepção do Ywyra. Foi necessário apenas criar 5 novos casos de uso, listados a seguir:

- Criar Requisição de Certificado de AC
- Excluir Requisição de Certificado de AC
- Alterar Requisição de Certificado de AC
- Exportar Requisição de Certificado de AC
- Concluir Criação de AC

Além dos casos de uso, novos artefatos de engenharia de software, como diagramas de atividades e diagramas de banco de dados, foram criados para facilitar a etapa de implementação efetiva do software.

4.1.1 *FUNCIONALIDADES*

A proposta foi aceita pelo ITI, e deu-se inicio a implementação do Ywyra. Com toda carga de conhecimento obtido no desenvolvimento do Ywapa e com a base tecnológica criada durante todo projeto, a implementação das novas funcionalidades para gerir uma AC Intermediária se deu de forma simples. Pela análise dos novos casos de uso, notou-se que todos faziam parte do processo de criação de Autoridade Certificadora, que na modelagem do sistema é a uma ação restrita ao perfil de administração. Portanto, todas as funcionalidades relativas ao perfil de operação poderiam ser reutilizadas do Ywapa, sem necessidade de alterações. Os esforços se concentraram então na área do código responsável pela criação de Autoridades Certificadoras, onde foram identificadas as funções que deveriam ser alteradas e quais novas funções deveriam ser criadas para o Ywyra.

A Autoridade Certificadora Intermediária, diferente da raiz, não possui um certificado auto-assinado, portanto para criá-la é preciso que outra AC emita um certificado para ela. Como

descrito no capítulo 2.3.5, para solicitar um certificado a uma AC, gera-se um par de chaves e uma Requisição de Certificado associada a chave pública do par de chaves gerado. Assim a lógica de criação de AC ficou da seguinte forma:

Iniciar criação de AC:

A primeira etapa da criação da AC é a geração do par de chaves e a criação da requisição de certificado com as informações da AC. A implementação da criação dessa requisição não é muito diferente da implementação da criação de um certificado auto-assinado, graças as facilidades proporcionadas pela Libcryptosec. Fazendo o uso do wizard de emissão de certificado, obtém-se as informações relativas a AC e cria-se a requisição, que é então inserida no banco de dados e em seguida só precisa ser exportada. Até aqui a AC ainda não está disponível para operação.

Finalizar criação de AC:

Com o certificado já emitido por uma outra AC em mãos, o administrador do Ywyra deve concluir a criação da AC. Isso é feito com a importação do certificado para o sistema. Nesse momento são realizadas algumas verificações para garantir a integridade do sistema, como a verificação da chave do certificado, que deve ser a mesma gerada na criação da requisição. Ao fim dessa operação a AC estará pronta para operação, e passará a estar habilitada na tela de seleção de perfil.

Além das funções para criação de AC, foram criadas funções de suporte, como exportação da requisição para arquivo, exclusão da requisição do sistema, e alteração dos dados de requisições já existentes, caso em que altera-se a requisição mantendo o mesmo par de chaves.

4.1.2 *MUDANÇAS NO BANCO DE DADOS*

A primeira vista, imaginou-se que seria necessário a criação de uma nova tabela no banco de dados, para armazenar as Requisições de Certificado das Autoridades Certificadoras. Porém, durante a implementação das mudanças no código, notou-se que a criação de uma tabela traria um custo muito alto para um problema pequeno, além de que não era necessário manter salvo o histórico das requisições que estiveram no sistema. Fazendo um novo estudo da modelagem do banco, definiu-se que a melhor solução seria fazer uso da coluna responsável por armazenar o certificado da Autoridade Certificadora. Ficou convencionado que enquanto o certificado da AC não fosse importado, a requisição ficaria armazenada nesse campo, e no momento que se efetivasse a importação, a requisição seria substituída pelo certificado da AC. Além dessa convenção, foi adicionada uma coluna com o estado da Autoridade Certificadora no sistema.

Esse campo identifica se a AC já está ativa ou não, o que depende de o certificado já ter sido importado ou não.

4.2 DIFICULDADES NO PROCESSO DE MANUTENÇÃO

A criação do novo projeto implicou na configuração de um novo ambiente de trabalho, parecido com o ambiente de desenvolvimento do Ywapa, com um Trac e um SVN. Conforme os projetos foram recebendo novas demandas de manutenção, percebeu-se que a maioria delas exigiam mudanças em partes que Ywapa e Ywyra possuíam em comum, já que o código de um era derivado do outro. Por estarem separados, essas mudanças causavam replicações de tickets e alterações idênticas em ambos os códigos. Além de que, nessa configuração de ambiente, era possível ocorrer a implementação de soluções diferentes para o mesmo problema, causando uma assincronia entre os códigos, e dificultando ainda mais todo o processo de manutenção.

Foi necessário repensar o ambiente de desenvolvimento dos projetos, para que fosse possível gerenciar suas partes em comum, sem problemas de sincronia de código e replicações de tickets.

4.3 UNIFICAÇÃO DOS CÓDIGOS

Com as dificuldades enfrentadas após a criação do novo projeto, iniciou-se um estudo para solucionar o problema de gerenciamento de partes em comum dos códigos. Encontrar uma forma de aplicar alterações uma única vez e tê-las efetivadas nos dois projetos era imprescindível, já que muito tempo era perdido na criação de tickets e na verificação e implementação de alterações nos dois códigos.

A solução veio com o uso de macros, que podem ser definidas e acessadas durante a compilação. Esse método permite a identificação, dentro de um arquivo, das linhas que serão interpretadas pelo compilador, através de estruturas semelhantes a clausulas ”if else”, usadas para verificar se uma macro foi definida ou não. Assim ficou possível mesclar os códigos do Ywapa e do Ywyra, e interpretá-los da forma correta, em tempo de compilação, apenas com a definição de uma macro.

Segue um exemplo para melhor entendimento do uso de macros. Trata-se de um código simples que imprime ”Hello world” ou ”Goodbye world”, dependendo da macro definida em sua compilação.

```
1 #include <iostream>
```

```
2
3  int main()
4  {
5  #ifdef Hello
6    std::cout << "Hello ";
7  #elif Bye
8    std::cout << "Goodbye ";
9  #endif
10   std::cout << "world." << std::endl;
11 }
```

Código Fonte 4.1: Exemplo do uso de macros

Dentro da estrutura ”#ifdef #elif” encontram-se as partes distintas dos códigos, a impressão das strings ”Hello” e ”Goodbye”. Fora da estrutura encontra-se a parte em comum dos dois códigos, a impressão da string ”world.”. No momento da compilação, define-se a macro que será usada, fazendo o compilador passar pelas partes desejadas do código. Para exemplificar, supõe-se um arquivo cujo nome seja exemploIFDEF.cpp que contém o código usado como exemplo anteriormente. A compilação através da linha de comando, com o uso do compilador g++[1] ficaria da seguinte forma:

$ g++ exemploIFDEF.cpp -DHello -o executavel

ou

$ g++ exemploIFDEF.cpp -DBye -o executavel

O parâmetro ”-D” pré-define uma macro, que será usada durante a compilação do arquivo. Neste caso, a primeira linha de comando pré-define a macro Hello. Isso faz com que o compilador ao passar pela cláusula ”#ifdef Hello” entre e compile o comando de impressão da string ”Hello” e ignore qualquer código que esteja dentro da cláusula ”#elif Bye”. Já a segunda linha de comando, que pré-define a macro ”Bye”, irá funcionar exatamente da forma inversa, ignorando o código dentro da clausula ”#ifdef Hello” e compilando apenas o código dentro da clausula ”#elif Bye”.

Esta solução tem efeitos colaterais, já que ela pode dificultar a legibilidade do código e acarretar em problemas de adaptação para eventuais novas equipes de desenvolvedores. No entanto entre as possíveis soluções, essa foi a de melhor custo beneficio encontrada pois o impacto no código não seria grande uma vez que não são muitas as diferenças de um software para o outro e o processo de união seria simples.

[1] http://gcc.gnu.org/

4.3.1 *IMPLEMENTAÇÃO*

Para implementar as mudanças no código, foi necessário encontrar uma forma simples para a comparação dos arquivos dos dois projetos. O SVN provê uma funcionalidade de sincronização de código, que permite a visualização das diferenças entre a versão do repositório e a versão atual na máquina. Aplicando a sincronização do repositório de um dos projetos nos arquivos do outro obtém-se o resultado desejado, ou seja, os arquivos e linhas de código que possuem diferenças de um projeto para outro.

O sequência de passos tomados para realizar esse processo de sincronização foi: copiar a última versão estável do Ywapa para um novo repositório SVN, fazer checkout dessa versão para uma máquina, substituir os arquivos baixados pelos arquivos da última versão estável do Ywyra e aplicar a sincronização. Através desse processo obteve-se todas diferenças entre os arquivos compartilhados pelo Ywapa e Ywyra, chegando a um total de 23 classes que deveriam ser alteradas, para que dessem suporte a ambos os projetos. As mais importantes estão listadas a seguir:

- UiController, SystemController e PersistenceController
- CertificateAuthorityStorageManager
- CertificationAuthorityBuilder e CertificationAuthority
- CertificateDataController e CertificateDataWizard
- XMLCertificateData
- LogData
- SignerThread

Algumas outras classes tiveram apenas alterações em strings que eram definidas no código, como por exemplo as mensagens que possuíam o nome Ywapa ou Ywyra no seu texto. Esses casos foram tratados através de uma constante com o nome do SGC, que assume o valor Ywapa ou Ywyra dependendo da escolha de sistema na compilação.

Além das alterações nas classes compartilhadas pelos dois projetos, duas novas classes que existiam somente no Ywyra foram adicionadas a esse novo projeto único. São as classes YwyraMainWindow e CertificateRequestStorageManager, das quais a primeira controla as funcionalidades presentes na janela principal do Ywyra, que são diferentes das funcionalidades da

janela principal do Ywapa, e a segunda é responsável pelo controle do salvamento e carregamento da requisição que fica salva no banco de dados, característica existente também somente no Ywyra.

Por fim, fez-se alterações na estrutura de pastas dos antigos projetos. O QT trabalha com arquivos no formato XML, os quais contém as informações de cada janela do software, que são interpretados durante a compilação, gerando assim arquivos chamados de base, que por sua vez são a implementação em C++ da interface. Como tornou-se necessário que a compilação provesse suporte aos dois projetos, criaram-se pastas separadas para esses arquivos XML e base, evitando possíveis conflitos durante a compilação do código.

Tendo um código compilável e executável, bastou definir a configuração do resto do ambiente de desenvolvimento. Poucas alterações foram necessárias, e um SVN e um Trac foram criados para o novo código. Os tickets reportados no Trac passaram a ter na sua descrição a informação do projeto a que pertenciam, e para alterações nas partes em comum do código um ticket simples, sem referência a nenhum projeto, passou a ser criado.

Com todas as alterações feitas, obteve-se um ambiente de desenvolvimento mais organizado e sem problemas de sincronia entre os projetos ou na utilização do Trac. Dessa forma foi possível continuar com o processo de manutenção, sem mais alterações no ambiente de desenvolvimento.

4.3.2 *DIFICULDADES NA IMPLEMENTAÇÃO*

A maior dificuldade encontrada durante a junção dos códigos foi com a pré-compilação feita pelo QT. Em umas das classes do Ywyra existia um slot (função ativada quando um sinal é disparado) que não é usado no Ywapa. Foi usada a cláusula ”#ifdef” para restringir a compilação desse slot, porém como a compilação é feita através de um compilador interno do QT a cláusula foi ignorada, fazendo com que o slot fosse compilado para o Ywapa. Tomando ciência desse comportamento, iniciou-se uma busca nas documentações do QT para descobrir como fazer uso de macros em sua pré-compilação. A resposta foi encontrada, mas não com a solução. A documentação do QT informava que não havia suporte a macros no compilador do QT, e que esse ignorava todas as cláusulas ”#ifdef #elif”. Para esse caso específico do slot do Ywyra, não havia problema em deixá-lo no código do Ywapa, pois não afetava nenhuma de suas funcionalidades. Porém, dessa forma, o código fica ”sujo” com código não utilizado, além de possibilitar a ocorrência de erros não previstos em futuras implementações. Nenhuma solução foi encontrada para esse problema, mas manteve-se o código da forma que estava, para quem sabe em uma futura atualização do QT, com suporte ao uso de macros, se fazer as mudanças necessárias.

5 MELHORIAS NO SGC

A implementação de melhorias no sistema SGC foi, de maneira geral, a principal atividade desenvolvida pelos autores durante a sua participação no projeto. Dentre essas melhorias encontram-se a correção de falhas, aprimoramento da usabilidade e principalmente a implementação de novas funcionalidades. Isso resultou de pequenas a grandes modificações que se estenderam por praticamente todas as partes do software. As melhorias de maior impacto e importância estão descritas nas seções a seguir.

5.1 ROTINA DE ATUALIZAÇÃO DE BASES DE DADOS

À medida que novas versões do software eram lançadas, fizeram-se necessárias algumas mudanças na estrutura das bases de dados, principalmente inclusão de novas colunas em algumas tabelas e alterações nas restrições, ou até a inclusão de novas tabelas. Nas versões iniciais isso não trouxe nenhum problema, pois estas apenas serviam para marcar etapas do desenvolvimento. No entanto, a partir do momento em que a primeira versão estável começou a ser utilizada pelo ITI, e com a constatação de que novas alterações nas bases de dados seriam necessárias na versão subsequente, surgiu a preocupação quanto a incompatibilidade entre backups gerados em diferentes versões do software. Se o ITI decidisse instalar o software mais atual, o procedimento mais prudente seria antes gerar um backup dos dados do sistema, para então proceder com a atualização, mas o backup gerado não poderia ser restaurado no novo software já que o esquema das bases restauradas seria inconsistente em relação ao sistema. O software fatalmente falharia após uma operação de restauração de backup nestas condições.

5.1.1 *ATUALIZAÇÃO DE BACKUP*

Como uma forma de solucionar o problema de incompatibilidade, foi proposta a implementação de um mecanismo capaz de detectar a versão em que um backup foi gerado e então fazer automaticamente as modificações nas bases de dados restauradas a fim de adequá-las à versão instalada. Para tornar isso possível, optou-se pela criação de um arquivo xml contendo a versão do

software, o qual deveria ser incorporado ao pacote de backup e lido no momento da restauração, possibilitando assim a averiguação da necessidade de atualização.

A implementação resultou na criação de duas novas classes, sendo uma delas um gerenciador de atualizações e a outra uma thread específica responsável pelo controle do processo. Com o gerenciador de atualizações, incluído no pacote de classes de persistência, ficaram as tarefas de leitura do arquivo xml e modificação das bases de dados. A criação do arquivo xml foi incluída no controlador de persistência.

Ainda durante a fase de programação da solução, percebeu-se que salvar simplesmente a versão do sistema não era uma boa abordagem, já que provavelmente haveriam versões sem modificações na estrutura das bases de dados, o que possivelmente viriam a expandir a quantidade de mapeamentos em relação às atualizações necessárias. Decidiu-se então pela criação de um versionamento específico para os backups, o qual iniciaria com o valor 1 e seria incrementado toda vez que uma nova versão do sistema fosse lançada contendo modificação nas bases de dados. Backups que não contivessem o arquivo xml, ou seja, geradas em um sistema sem o mecanismo de atualização, seriam considerados versão 0.

Depois de finalizado, o mecanismo de atualização de backup funcionava da seguinte maneira: Durante o processo de restauração do backup, logo após a restauração das bases de dados, o controlador de sistema instanciava a thread de atualização, passando-lhe o controle sobre essa operação. A thread por sua vez chamava o controlador de persistência que instanciava o gerenciador de atualização e chamava o método responsável por efetuar as modificações de fato.

Na instanciação do gerenciador de atualização ocorria o carregamento do arquivo xml, e o valor lido era guardado num atributo da classe. Em seguida, quando o método de atualização era chamado, este verificava o valor do atributo de versão para determinar quais modificações precisavam ser realizadas. O método de atualização consistia simplesmente de um switch sem breaks, com um case para cada versão de backup existente, dentro do qual havia uma chamada para o método que implementava as modificações relativas àquela versão. Veja exemplo (c.f 5.1):

```
1 switch( versao_backup )
2 {
3     case 0:
4             atualizeParaVersao1();
5     case 1:
6             atualizeParaVersao1();
7     ...
8
```

```
9       case N:
10                  atualizeParaVersaoN();
11                  break;
12  }
```

Código Fonte 5.1: Exemplo da sequência de atualização

Assim o método saltava direto para as modificações respectivas à versão do backup e seguia promovendo todas as modificações referentes às versões subsequentes, até alcançar a atual.

5.1.2 *ATUALIZAÇÃO DE BASES DE DADOS*

Tão cedo que iniciaram-se os testes da rotina de atualização de backup ficou evidente que um aspecto importante do problema havia passado despercebido. O fato é que ele não se manifestava apenas na operação de restauração de backup. O próprio ato de instalar uma nova versão do software em um ambiente previamente em operação já iria acarretar incompatibilidade entre as estruturas de bases de dados, pois o sistema tentaria acessar aquelas bases criadas com a versão antiga como se tivessem a estrutura atual, inevitavelmente causando várias falhas. Com isso constatou-se que a implementação inicial da rotina de atualização teria que ser repensada, pois o uso de um arquivo xml para guardar a informação de versão, agora não mais somente no contexto do pacote de backup, não parecia ser a alternativa mais indicada, uma vez que seria necessário garantir que esse arquivo não sofresse alterações externas.

Esta situação levou em primeiro lugar a uma mudança conceitual, já que não se tratava mais de um versionamento dos backups, mas sim de um versionamento da estrutura das bases de dados, ou como foi chamado internamente, versão de bd. E para implementar isso optou-se por abandonar o xml e passar a guardar a informação de versão no próprio banco de dados, através de um atributo adicional na tabela de configurações do sistema. Agora toda vez que o sistema é iniciado ou um backup é restaurado ocorre a comparação entre a versão de bd salva no banco e o valor da constante *versão de bd* inserida diretamente no código fonte. Se o valor do banco for menor que o da constante a rotina de atualização é iniciada.

Como o valor de versão é salvo na tabela de configurações do sistema não há necessidade de carregá-lo separadamente, uma vez que estas configurações já são carregadas logo que se inicia o sistema, ou no caso do backup assim que as bases da dados são restauradas. As duas classes criadas anteriormente foram mantidas, sofrendo apenas algumas alterações, já que ao invés de ler o valor da versão a partir do arquivo dentro do construtor do gerenciador de atualização, este valor agora é acessado pelo controlador do sistema e passado como parâmetro para a thread, que o manda para o gerenciador de atualização através do controlador de persistência (Figura 3). Ao

final do processo de atualização, se ela for realizada com sucesso, o valor da constante *versão de bd* é passado para as configurações do sistema. No método que carrega as configurações de sistema foi preciso adicionar uma exceção para o caso onde a coluna de versão de bd não exista na tabela, quando define-se que a versão das bases de dados é 0, ou seja, bases criadas antes da introdução do mecanismo de atualização.

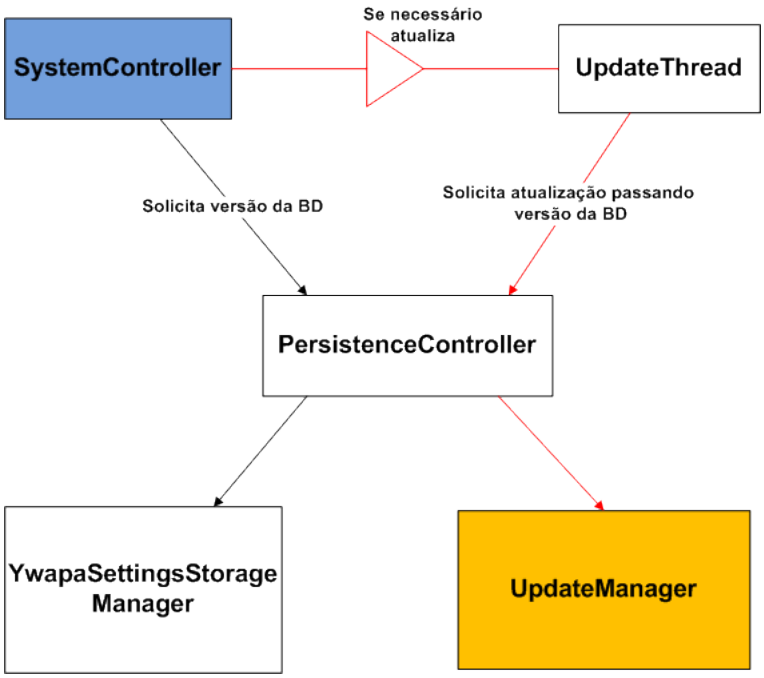

Figura 3: Repasse de controle para atualização

O método que define quais modificações devem ser realizadas para atualizar determinada versão permaneceu inalterado.

5.2 SUPORTE A NOVOS ALGORITMOS

O NIST recomenda que o algoritmo de hash SHA-1, o mais usado atualmente, tenha seu uso interrompido até 2011 [10]. Isso significa que, do ponto de vista deste órgão, todos os documentos eletrônicos que fizerem uso do algoritmo SHA-1 estarão com sua verificabilidade sob suspeita após essa data limite. Essa eminência para o fim da validade do SHA-1 afetou diretamente o SGC, que trabalhava unicamente com esse algoritmo. Por demanda do ITI, iniciou-se o trabalho de implementação do suporte de novos algoritmos de hash da família SHA-2, e além disso do suporte ao uso de chaves ECDSA, baseadas em curvas elípticas.

5.2.1 *SHA-2*

A família SHA-2 é constituída pelos algoritmos SHA-224, SHA-256, SHA-384 e SHA-512, onde o número identifica o tamanho em bits do resumo criptográfico gerado pelos algoritmos. O SHA-1, suportado até então, é de 160 bits.

A demanda para o suporte aos novos algoritmos implicou principalmente em mudanças na biblioteca criptográfica Libcryptosec, que suportava apenas os algoritmos MD5 e SHA-1. Como descrito no capítulo 3, a Libcryptosec é uma abstração para o paradigma de programação de orientação a objetos de algumas funcionalidades providas pelo OpenSSL, que é desenvolvido em puro C estruturado. Portanto, a parte mais complexa da implementação se deu no código da Libcryptosec, exigindo um estudo na biblioteca do OpenSSL em busca das funções que realizavam as novas operações de resumo criptográfico e a inserção delas na Libcryptosec.

No SGC, de forma geral, bastou adicionar telas para a seleção do algoritmo de hash a ser usado na assinatura, e possibilitar que a escolha seja repassada à Libcryptosec. Essas telas aparecem na emissão de certificados e LCR e, no caso do Ywapa, na criação da AC.

5.2.2 *ECDSA*

O ECDSA é um algoritmo de assinatura digital que trabalha com curvas elípticas (2.2.2). Ele traz algumas vantagens sobre os algoritmos mais comuns como o RSA e DSA. Suas chaves, por exemplo, proporcionam uma maior segurança em comparação a chaves de mesmo tamanho no RSA ou no DSA, sem falar que seu processo de assinatura costuma ser significativamente mais rápido do que o dos demais algoritmos.

Da mesma forma que a implementação do suporte a família de hash SHA-2, as mudanças mais complexas para prover o suporte a ECDSA tiveram que ser feitas na biblioteca criptográfica Libcryptosec. O OpenSSL, que é a base da biblioteca, passou a dar suporte ao ECDSA somente em sua versão 1.0, que na época ainda estava em fase beta, enquanto a Libcryptosec fazia uso da última versão estável, a 0.9.8. O trabalho de migrar para essa nova versão teve um impacto muito maior do que o esperado no sistema, já que além da alteração de algumas funções, a migração trouxe dificuldades na própria compilação do SGC. Essas dificuldades são descritas na subseção 5.2.3.

No contexto do SGC, durante a criação da AC, adicionou-se a opção de uso do ECDSA, que ao ser selecionada, faz o campo de seleção do tamanho da chave, usado para o RSA e DSA, ser substituído por um campo com as curvas suportadas pelo sistema. Outra alteração necessária, que não havia sido prevista, foi a filtragem dos algoritmos de hash, devido a possibilidade do

uso de chaves ECDSA menores que alguns dos resumos criptográficos suportados e ao fato de o tamanho da chave usada na assinatura ter que ser necessariamente maior ou do mesmo tamanho do resumo criptográfico gerado. A aplicação do filtro é simples: Após gerar a chave ou carregá-la do MSC, obtém-se o seu tamanho e compara-o aos tamanhos dos hashes existentes. Os que forem menores ou iguais ao tamanho da chave são adicionados a uma lista que é repassada para a interface, de forma que somente os hashes suportados para aquela chave serão exibidos ao usuário.

5.2.3 *DIFICULDADES*

A maior dificuldade encontrada na implementação do suporte aos novos algoritmos foi a necessidade de migrar para uma versão mais nova do OpenSSL. A biblioteca criptográfica Libcryptosec faz uso de outra biblioteca chamada libp11[1], responsável pelo suporte a smart cards, a qual faz uso do OpenSSL 0.9.8 para implementar suas funções. Dessa forma criou-se uma incompatibilidade entre versões do OpenSSL, já que a LibCryptosec utiliza a versão 1.0 beta.

Como as bibliotecas eram usadas dinamicamente, ou seja, carregavam suas funções em tempo de execução, o SGC acabava encontrando problemas no processo de carregamento de bibliotecas necessárias para a libp11 e Libcryptosec, ocasionando falhas de segmentação. A solução para tal problema foi compilar estaticamente a Libcryptosec com o OpenSSL 1.0.0 e manter a libp11 dinâmica com o OpenSSL 0.9.8. Dessa forma, bastava ter o OpenSSL 0.9.8 instalado, pois as funções do 1.0.0 usadas na Libcryptosec já estavam inseridas estaticamente no código. O mesmo procedimento teve que ser feito na engine do MSC, que é baseada no OpenSSL e portanto também precisava fazer uso da versão mais nova para dar suporte ao ECDSA.

5.3 INDEPENDÊNCIA DE MSC

O projeto inicial do SGC previa o suporte ao uso de um Módulo de Segurança Criptográfico (MSC), responsável por armazenar de forma segura a chave privada da Autoridade Certificadora. Era previsto também que o sistema deveria ser compatível com vários modelos de MSC, para não ficar dependente de apenas um fabricante desses módulos, possibilitando a troca de um modelo antigo por um modelo mais novo e seguro, sem necessidade de alterações no código fonte do software.

[1] http://www.opensc-project.org/libp11/

Para prover tais funcionalidades, foi utilizado uma interface de comunicação entre a aplicação e o MSC, também conhecida por engine. Mais especificamente, foi utilizada a engine OpenSSL, devido a sua popularidade entre os sistemas de segurança. Dessa forma, o SGC poderia se comunicar com qualquer MSC, desde que esse tivesse também suporte a engine OpenSSL.

5.3.1 *TESTES COM O MSC LUNA*

Em torno do final de abril e início de maio de 2008 chegou ao laboratório um MSC Luna da SafeNet para que fosse feita a análise de compatibilidade do equipamento com o SGC Ywapa. Ele possuía uma engine OpenSSL implementada, portanto era esperado que seu uso se desse da mesma forma que o MSC utilizado no LabSEC. Porém, durante os testes realizados, foi preciso fazer uma configuração na engine para alterar o caminho para um determinado arquivo utilizado durante sua execução. Tal funcionalidade não existia no Ywapa, e implicava em alterações internas na engine ou em carregamento externo da engine através de comandos do OpenSSL.

O SGC, até então, tinha um suporte limitado em relação a engine do MSC. Era preciso configurar a engine previamente com informações como por exemplo o endereço IP do MSC que seria utilizado, o que dificultava eventuais trocas de hardware pois exigia uma alteração interna, no próprio código fonte da engine, caso esse IP viesse a ser modificado. Para um usuário comum, um processo desse tipo pode não ser viável. Além disso, outros modelos de MSC, como o Luna, poderiam exigir outros comandos de configuração para funcionarem corretamente. Isso motivou a implementação de melhorias no suporte à engine, tornando mais simples a sua configuração por parte do usuário e garantindo maior flexibilidade em relação a outros modelos de hardware. A próxima subseção descreve as mudanças que vieram a ser feitas para implementar essas melhorias.

5.3.2 *SUPORTE A COMANDOS PARA A ENGINE DO MSC*

O OpenSSL possibilita a atribuição de comandos de configuração à engine no momento do seu carregamento. Esses comandos são passados no formato de um par de valores, que identificam o comando e o parâmetro desse comando. Por exemplo, para configurar o IP do MSC com o qual a engine deve trabalhar passa-se, durante o carregamento dela, o comando ADDRESS_CONN e o endereço IP. Durante o tempo em que a engine permanecer carregada, ela trabalhará especificamente com o MSC identificado por este endereço IP.

Para utilizar essa funcionalidade no SGC foi preciso primeiro prover o suporte a ela na Libcryptosec. Nas classes de abstração da engine foram incluídas funções que faziam a atribuição

de comandos, no formato: void setCommand(command, value).

Feita essa etapa, pôde-se iniciar as alterações necessárias no SGC, que envolveram alterações na interface, core e persistência, listadas a seguir:

- **Persistência**

  As alterações necessárias na persistência foram a criação de uma nova tabela na base de dados para armazenar as configurações da engine, juntamente com os métodos para sua leitura e atualização. Essa tabela contém um id, uma coluna para o identificador do comando e outra para o valor passado como parâmetro.

- **Interface**

  Na interface foi necessário possibilitar a inserção dos comandos desejados e seus parâmetros na tela de configuração do MSC. Isso foi feito através de uma lista que pode ser incrementada com um par de valores, comando e parâmetro, e que possibilita também a remoção de pares existentes. Quando concluída a configuração, a interface repassa os dados para o core que por sua vez repassa para a persistência fazer as alterações necessárias no banco de dados. Essa tela de configuração pode ser acessada durante a inicialização do sistema ou então através do perfil de administração, que tem permissão de reconfigurar o MSC.

- **Core**

  Agora, se o sistema está configurado para usar um MSC, no carregamento da engine são executados também os comandos para a configuração desta, e para isto é feito uso das funções implementadas na Libcryptosec.

5.4 REGISTRO DE EVENTOS PARA AUDITORIA (LOGS)

O SGC opera em uma infraestrutura de enorme importância, pois é nela que se baseia a segurança de grande parte das transações financeiras eletrônicas, a validade de contratos e outros documentos eletrônicos, o sigilo de dados confidenciais e muito mais. Essas características exigem que o sistema seja auditável, ou seja, que seja possível a um determinado grupo de pessoas, os auditores, examinar a posteriori as operações realizadas nas ACs gerenciadas pelo software afim de garantir que está tudo de acordo com as restrições estabelecidas.

Para suprir a necessidade de auditoria, o SGC possui um mecanismo de registro de eventos, ou simplesmente logs, que guarda as informações sobre as operações realizadas, mesmo as que falham. Os logs são organizados por tipos de eventos, e estes por sua vez são divididos em

tipos de ações. Exemplos de tipos de eventos são: Evento de Backup, Evento de Certificado, Evento de AC, etc. Alguns tipos de ação, por exemplo para um Evento de Backup, são: geração, restauração, falha na geração, etc.

Um log completo possui os campos: data e hora (time_t); evento (código); ação (código); informações adicionais (string), as quais são opcionais e variam dependendo do tipo de evento e ação; e id do perfil. O campo data e hora é referente ao momento em que a operação é finalizada e o campo id do perfil indica se a operação foi realizada pelo perfil de administração ou de operação. Ações são classificadas ainda como normais ou críticas. Representação de estrutura do log na Figura 4.

Esses registros podem ser exportados. Para isso ocorre a geração de um arquivo xml que é assinado e exportado no formato PKCS#7. Os logs são utilizados também na geração dos relatórios gerenciais, como descrito na subseção 3.1.3.

Durante o processo de manutenção do software e implementação de melhorias foram incluídos alguns novos tipos de eventos e ações, bem com algumas modificações nos existentes, mas merecem destaque as modificações realizadas no intuito de corrigir as falhas na tradução dos logs na mudança de idioma do sistema.

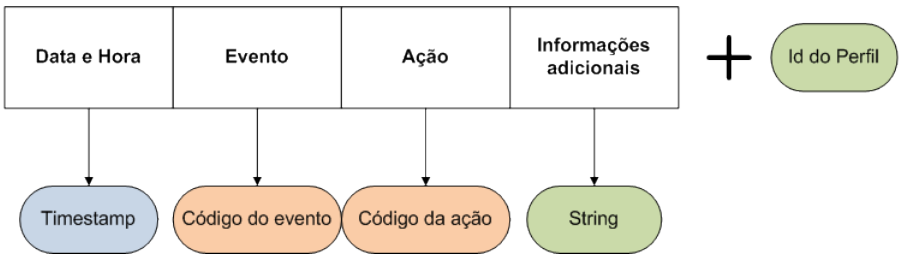

Figura 4: Estrutura do registro de evento

5.4.1 *CORREÇÕES NA TRADUÇÃO DE LOGS*

Apesar de o código fonte e as mensagens de interface do SGC terem sido escritos no idioma Inglês, sua interface suporta diferentes idiomas. Até o momento o sistema foi localizado apenas para o Português Brasileiro, mas por utilizar os mecanismos de internacionalização do QT, um novo idioma pode ser incluído com a simples criação de um arquivo de traduções, sem a necessidade de se fazer alteração alguma no código do software. Para que isso funcione todas as strings de interface são envolvidas com o método tr() do QT, e quando é solicitada a alteração de idioma, mesmo durante sua execução, esse método garante a tradução das strings exibidas para o idioma selecionado. O problema é que apesar de este mecanismo funcionar muito bem na maior parte do sistema, com os logs o comportamento não era o esperado. Tanto nas telas

de logs, quanto na exportação de relatório, que também faz uso dos logs, só era efetivada a mudança de idioma se o sistema fosse reiniciado, enquanto nas outras áreas isto ocorria imediatamente. A solução acabou por ser bastante simples, mas para descobrir o que estava causando o problema foi necessário gastar algum tempo estudando a fundo o funcionamento do mecanismo de registro de eventos, já que este havia sido implementado antes da mudança na equipe de desenvolvimento. Como mencionado anteriormente, os logs são armazenados na base de dados na forma de códigos, de forma que para exibi-los na interface ou incluí-los nos relatórios é usada uma tabela que faz o mapeamento do código para a string respectiva. Constatou-se que esta tabela é gerada dinamicamente, somente quando se inicia o sistema. Isso era o motivo de os logs não serem traduzidos na mudança de idioma, pois a tabela já estava montada com as strings do idioma em uso no momento de sua criação. Uma vez sob posse dessas noções, foi preciso somente introduzir uma nova chamada à função que gera a tabela de mapeamento de logs no momento da mudança de idioma.

5.5 OUTRAS MELHORIAS

5.5.1 *CORREÇÕES*

Essa subseção trata das principais correções feitas no SGC. Por correção, entende-se regras de negócios implementadas de forma incorreta, erros na lógica de programação e na modelagem do sistema além de erros relacionados a linguagem de programação.

CORREÇÕES NA INTERFACE GRÁFICA

Uma atividade quase constante durante esta fase de manutenção e melhoria do SGC foi a realização de correções na interface gráfica. Praticamente todas as funcionalidades do sistema que fazem uso da interface necessitaram de alguma atenção nesse sentido, desde algum simples redimensionamento a inclusão de novas telas ou botões, reposicionamentos, ajustes nas mensagens ou no texto básico, e inúmeras outras alterações. Algumas dessas correções foram motivadas por falhas de funcionamento, mas a grande maioria foi feita no intuito de melhorar a usabilidade, deixando o sistema o mais simples e prático possível.

VERIFICAÇÃO DE ASSINATURA DE REQUISIÇÕES

Conforme a RFC-2986 [18], a Autoridade Certificadora deve, antes da emissão do certificado, verificar a autenticidade da entidade requerente e validar sua assinatura. Isso é feito

através da chave pública e da assinatura presentes na requisição. Até então essa verificação não existia no SGC, que permitia até mesmo a emissão de certificado para requisições não assinadas. Esse comportamento foi alterado para que seja feita a verificação da assinatura, e quando não válida ou não existente, seja cancelada automaticamente a operação, com avisos ao usuário. Foram implementadas duas novas funções na Libcryptosec, uma para verificar se a requisição está assinada e outra que faz a verificação da assinatura. No SGC, bastou fazer uso dessas duas funções no momento da emissão do certificado, fazendo com que o processo terminasse caso alguma das verificações falhasse.

5.5.2 *MELHORIAS*

Essa subseção trata das principais melhorias implementadas no SGC. Por melhoria entende-se novas funcionalidades que têm a finalidade de melhorar a usabilidade do sistema, deixando-o mais simples e prático, prevenindo alguns erros de operação por parte dos usuários.

COMPARAÇÃO DOS DADOS DE UM CERTIFICADO E UMA REQUISIÇÃO

Como descrito na seção 2.3.5, as informações em uma requisição não são definitivas. Elas estão passíveis de alteração no momento da emissão do certificado, ficando a critério da AC os dados que devem ser adicionados, removidos ou modificados. Por esse motivo, quando foram feitos os primeiros testes no Ywyra, notou-se que seria interessante identificar, no momento da importação de um certificado para conclusão de criação de AC, campos que pudessem ter sido alterados na emissão. Já estava sendo exibido para o usuário um visualizador de certificado, que continha as informações do certificado importado, porém nenhuma comparação era feita com a requisição da AC. Definiu-se então que, através do uso do visualizador de certificados, seria feita uma marcação dos campos que foram adicionados, removidos ou alterados, com distinção por cores e formatação de texto para cada caso.

O processo de implementação dessa funcionalidade se deu em três partes específicas do projeto e estão descritas a seguir:

- **Interface**

  O QT na sua versão 3.3 dispõem de duas classes para a exibição de uma árvore de dados. Essas classes são QListView e QListViewItem, onde a QListView representa o campo onde a árvore é montada, e a classe QListViewItem é responsável por armazenar cada item da árvore, junto com informações como o item pai e os itens filhos desse QListViewItem. Infelizmente essas classes não possuem nenhuma funcionalidade de formatação do texto,

e como o intuito da solução era apresentar através de diferentes cores os campos modificados, foi preciso fazer uma especialização da classe QListViewItem, para que essa desse suporte a formatação de texto. A nova classe possuía um atributo com a cor que o item deveria possuir e três atributos booleanos, que definiam se o texto estava em negrito, itálico ou riscado. Além desses atributos, a função paintCell da classe QListViewItem foi reimplementada para que construísse o item conforme definido nos atributos. Essa reimplementação foi feita conforme o código 5.2.

```
1  void MyListViewItem::paintCell( QPainter *p, const QColorGroup &cg,
2                                  int column, int width, int
                                       alignment )
3  {
4          QColorGroup _cg( cg );
5          QColor c = _cg.background();
6          _cg.setColor( QColorGroup::Text, this->fontColor );
7
8          QFont font = p->font();
9          font.setBold(this->bold);
10         font.setItalic(this->italic);
11         font.setStrikeOut(this->strikeOut);
12         p->setFont(font);
13
14         QListViewItem::paintCell(p, _cg, column, width, alignment);
15
16         _cg.setColor(QColorGroup::Text, c);
17 }
```

Código Fonte 5.2: Reimplementação da função paintCell

- **libcryptosec**

  O formato do XML construído pela Libcryptosec até então não estava seguindo um padrão bem definido, o que causou algumas dificuldades para a implementação da comparação entre certificado e requisição, principalmente entre os valores das extensões do certificado. A biblioteca, a princípio, só considerava na construção do XML as extensões conhecidas por ela, impedindo a comparação de valores de extensões desconhecidas. A estrutura em que XML se encontrava é exibida no código fonte 5.3.

```
1 <basicConstraints>
2    <extnID>basicConstraints</extnID>
3    <critical>yes</critical>
```

```
4     <extnValue>
5         <ca>true</ca>
6         <pathLenConstraint>-1</pathLenConstraint>
7     </extnValue>
8 </basicConstraints>
```

Código Fonte 5.3: Formato antigo do XML para extensões

Verificou-se que a tag XML que iniciava uma extensão deveria conter o nome da extensão, o que não era possível no caso de extensões desconhecidas, onde só se tem o valor do seu OID. Tendo conhecimento de que uma extensão sempre terá o conjunto de valores: OID, IS_CRITICAL e VALUE, a seguinte proposta (c.f. 5.4) de alteração no XML foi feita.

```
1 <extension>
2     <extnID>basicConstraints</extnID>
3     <oid>2.5.29.19</oid>
4     <critical>yes</critical>
5     <extnValue>
6         <ca>true</ca>
7         <pathLenConstraint>-1</pathLenConstraint>
8     </extnValue>
9 </extension>
```

Código Fonte 5.4: Novo formato do XML para extensões

Nesse formato toda extensão é iniciada com a tag extension, e então dentro dessa tag se tem as informações relativas a extensão, como o OID, se ela é crítica ou não, o seu valor, e caso seja uma extensão conhecida o seu nome. Essa proposta foi aceita e implementada como uma nova função, e por razão do legado de uso da biblioteca, manteve-se a função antiga de geração de XML.

- **Core**

  No SGC, foram criadas as funções responsáveis por fazer a comparação entre os XMLs. No momento da importação de um certificado para conclusão da criação da AC, obtém-se o XML do certificado importado e da requisição presente no banco de dados, e com eles realiza-se uma busca de um campo de um nos campos do outro, para identificar onde foram feitas alterações. Durante essa comparação um novo XML é montado e quando uma diferença é encontra marca-se a tag do campo modificado, identificando se a mudança se trata de uma adição, remoção ou alteração. Com esse novo XML marcado, inicia-se o processo de montagem da árvore de exibição de dados. Fazendo uso da classe

implementada, especialização de QListViewItem descrita anteriormente, e dependendo de como o campo do XML está marcado, atribui-se um valor de cor e formatação para o item da árvore. Por exemplo, texto riscado e na cor vermelha para os itens removidos.

A convenção adotada para a formatação e a coloração dos campos foi a seguinte: Na cor verde em negrito os campos que foram adicionados; vermelho em negrito e riscado os campos removidos; e por fim, amarelo e em negrito os campos com seus valores alterados. Dessa forma a solução ficou concluída, com a exibição clara e simples das diferenças entre o certificado importado e a requisição que havia sido criada para a AC, conforme a figura 5.

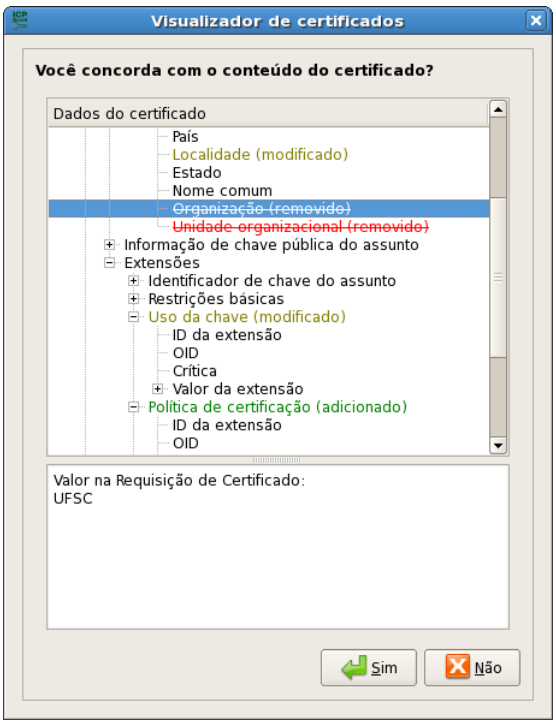

Figura 5: Comparador de certificado e requisição

AVISOS DE OPERAÇÕES NÃO RECOMENDADAS

Existem no SGC algumas operações que devem ser possíveis de serem realizadas, mas que não são recomendadas pelas boas práticas. Por exemplo, toda LCR possui em seus campos um intervalo de validade, e é uma boa prática que não se deixe existir mais de uma LCR válida ao mesmo tempo (seção 2.3.2). Por esse motivo, para se evitar que LCRs sejam emitidas dentro do prazo de validade de outras, implementou-se logo antes da emissão de uma LCR uma verificação da existência de outra LCR ainda válida. Se alguma for encontrada, é exibida uma mensagem de aviso de sua existência, evitando assim que o usuário do SGC faça essa emissão por engano. Além da verificação da LCR, existem também algumas verificações na emissão de

certificados. Não é comum a emissão de certificados com o mesmo common name ou com o mesmo par de chaves, mas apesar de não ser comum deve ser possível realizar essas operações, pois alguns requisitos da ICP-Brasil, como a renovação de certificados, implicam justamente na emissão de certificados com mesmo common name e chave. Porém, novamente para se evitar um possível engano do usuário do SGC, foram adicionadas verificações da existência de um certificado já emitido com os mesmos dados de common name e chave pública de um certificado que está para ser emitido, e quando encontrado, uma mensagem de aviso é lançada, para que o usuário tome conhecimento da existência do outro certificado.

RESTRIÇÕES EM SMART CARDS

O SGC desde o inicio trabalhou com a autenticação de usuário através de smart cards, no entanto essas autenticações sempre foram feitas de maneira simples, através da comparação do certificado no cartão com o certificado cadastrado no banco de dados. Durante o tempo de manutenção do software, cogitou-se a possibilidade da melhoria desse processo de autenticação, com o uso de verificações mais restritivas e que impediriam possíveis fraudes. A princípio pensou-se em uma solução ideal, com verificação de validade de certificado através de cadeias de certificação e LCR, mas logo viu-se que essa solução seria complexa demais para um problema não tão sério assim. Então decidiu-se apenas impedir o cadastro e autenticação de usuários com certificados expirados, ou seja fora da data de validade. Essa verificação é simples e é feita através da comparação do campo notafter do certificado com a hora atual do sistema, porém a implementação dessa restrição trouxe um novo problema, pois o usuário pode não estar ciente de que seu certificado está para expirar, e acabar não cadastrando um certificado mais novo, o que acarretaria num bloqueio do sistema, já que ele não conseguiria mais se autenticar. Para prevenir essa situação, foram adicionados alertas para quando for identificado um certificado com menos de seis meses de validade, possibilitando dessa forma que o usuário previna, com um bom tempo de antecedência, o bloqueio de sistema.

ALTERAÇÃO DOS DADOS DA REQUISIÇÃO NA EMISSÃO DE CERTIFICADO

Na proposta inicial do SGC, não era previsto a alteração dos dados de um requisição, apenas a inclusão de extensões, feita durante a emissão do certificado. Conforme se realizavam testes no sistema, verificou-se que, em alguns casos, uma requisição podia ter algum valor errado, devido a erro de digitação por exemplo, o que tornava necessária a criação de outra requisição, atrasando assim todo o processo. Com uma visão mais crítica, imaginou-se os problemas que isso acarretaria se um erro na requisição fosse encontrado durante uma cerimonia oficial de

emissão de certificado, que normalmente exige muito tempo e dinheiro para ser organizada. A solução seria possibilitar a edição dos campos que vinham na requisição de certificado, e inclusive as extensões. No entanto, até o momento apenas os campos do Assunto da requisição eram lidos e exibidos na forma de um xml na primeira tela do wizard de emissão. As extensões por ventura presentes eram ignoradas. Para por a solução em prática primeiramente foi removido o xml e usado o formulário de preenchimento de campos de Assunto, deixando a tela inicial do wizard igual a da operação de criação de AC. Em seguida foi alterado o procedimento de carregamento da requisição, de forma que todos os campos do assunto e as extensões presentes passaram a ser identificados e os seus valores usados para popular automaticamente o wizard, possibilitando assim a sua edição por parte do usuário. Foi criada ainda uma tela adicional que lista os OIDs de extensões presentes mas que não são suportadas pelo sistema, dando ao usuário a possibilidade de removê-las se assim desejar.

SUPORTE AO FORMATO DER

Existem dois formatos de arquivos que são usados pelo Openssl, são o formato binário ASN1 DER e o formato PEM, que faz uso da Base64 e cabeçalhos de início e fim de arquivo. O SGC implementava apenas exportação e importação de arquivos nesse segundo formato, o PEM, e para uma maior compatibilidade decidiu-se implementar o suporte ao formato DER. As soluções tomadas para cada caso, exportação e importação, estão descritas a seguir.

- **Importação**

  A primeira proposta de diferenciação entre os formatos na importação do arquivo foi através do cabeçalho existente no PEM. Se fosse encontrado o cabeçalho na leitura do arquivo consideraria-se que o arquivo estava no formato PEM, se não, estaria no formato DER. Porém essa proposta não foi aceita, devido a complicações com leitura do arquivo, e por aparentar não ser uma solução elegante. A segunda proposta, que foi aceita e implementada, foi considerar que o arquivo sempre estaria no formato PEM, e caso não estivesse, uma exceção seria lançada, fazendo então que se tentasse fazer a leitura do arquivo no formato DER. Se o arquivo não fosse de nenhum dos formatos, uma segunda exceção seria lançada e uma mensagem de formato de arquivo inválido seria exibida. Dessa forma o suporte a importação de arquivos nos dois formatos ficou transparente para o usuário final, sem a necessidade da escolha do formato do arquivo durante a importação.

- **Exportação**

  A solução para a exportação de arquivos em DER foi mais simples que a importação.

Através da extensão do arquivo, fez-se a definição do formato que ele deveria ser exportado, caso não fosse uma extensão conhecida o formato PEM seria usado. Para se obter os dados no formato DER, bastou fazer uso da funcionalidade de conversão dos dados de PEM para DER da Libcryptosec . Concluiu-se assim a exportação de arquivos no formato DER.

5.5.3 *ADEQUAÇÕES*

Essa subseção trata das principais adequações implementadas no SGC. Por adequações entende-se ajustes feitos no sistema para que ele estivesse de acordo com as recomendações das RFCs e às necessidades do ITI e da ICP-Brasil.

FLEXIBILIZAÇÃO NA ADIÇÃO DE PONTOS DE DISTRIBUIÇÃO DE LCR E DE IDENTIFICADORES DE POLÍTICAS

Nas primeiras versões do SGC, na Criação de ACs e emissão de certificados, era possível a adição de apenas um caminho para a extensão Pontos de Distribuição de LCR. O mesmo acontecia com os Identificadores de Políticas. Em contrapartida, a ICP-Brasil requeria a possibilidade da adição de mais caminhos e identificadores respectivamente. Para resolver esta incompatibilidade foi introduzida mais uma etapa no wizard de emissão de certificados, o qual é usado também na criação de AC, destinada especificamente para a edição das duas extensões citadas. Agora é possível incrementar indefinidamente as listas de Pontos de Distribuição de LCR e de Identificadores de Políticas.

RESTRIÇÕES IMPOSTAS PELAS RFCS

As funcionalidades de criação de AC e emissão de certificados suportam por meio de interface gráfica inclusão da maioria, ou pelo menos a parte mais usada, dos tipos de campos e extensões de certificados presentes na especificação do padrão X509, deixando o usuário livre para escolher quais desses campos e extensões deseja preencher. No entanto, a RFC possui algumas restrições em relação à combinação de algumas extensões. Por exemplo, se na extensão Basic Constrains o campo CA tiver o valor TRUE, indicando que se trata de um certificado de AC, é obrigatória a inclusão da extensão Identificador de Chave do Assunto. Para esse caso, e outros similares, foram incluídas verificações na interface de forma que se uma extensão é incluída a outra relacionada é incluída automaticamente. Se o usuário não concordar ainda pode remover esta segunda, o que gera uma mensagem de alerta, mas o prosseguimento da emissão não é impedido. De forma similar, quando uma requisição de certificado é importada durante a

operação de emissão é feita a verificação de seu conteúdo e inclusão automática das extensões necessárias para cumprir as restrições. O usuário é avisado sobre as modificações realizadas.

DESATIVAÇÃO DE MEMBROS DE GRUPOS

Afim de manter um histórico dos membros que participaram de algum grupo de perfil, foram propostas as seguintes alterações na lógica de edição de grupos:

- Não permitir a exclusão de membros efetivados no grupo.
- Adicionar a opção de desativação e ativação de membro.

Dessa forma, um membro que passou a fazer parte de um grupo sempre fará parte dele, impedindo assim a perda de suas informações, e possibilitando um rastreamento desse membro a qualquer momento que for preciso. O impacto no sistema com a implementação dessas duas mudança foi de certa forma grande, já que exigiu alterações em diversos pontos do código, descritos a seguir.

- **Persistência**

  No banco de dados foi adicionada uma coluna com o estado do membro no grupo, ativo ou inativo, e a coluna que mantém a parte do segredo compartilhado passou a aceitar valores nulos, já que membros inativos não fariam parte do compartilhamento de segredo.

- **Interface**

  Foram discutidas maneiras de se exibir de forma clara os membros que estão ativos, inativos ou que acabaram de ser adicionados no grupo. Decidiu-se por fazer uso da mesma solução utilizada para exibir a diferença entre um certificado e uma requisição (5.5.2). Com o uso da classe especializada de QListViewItem, ficou possível formatar as informações na lista de membros, de forma a exibir os membros inativos na cor cinza. Os membros recém adicionados em negrito e os membros ativos com a fonte normal do sistema. Além dessa distinção por cores, foi acrescentado um campo de seleção, que permite definir se membros não ativos devem ser exibidos ou não, evitando assim a listagem de todos membros que já participaram do grupo. Foi adicionado também um botão para ativação e desativação de membros, e o botão de exclusão passou a ficar habilitado somente quando o membro acabou de ser inserido no grupo e enquanto a operação ainda não foi salva.

- **Core**

  No core do sistema foram necessárias alterações na lógica do algoritmo de compartilhamento de segredo, que até então dividia o segredo entre todos os membros do grupo. Com as novas mudanças, o compartilhamento passou a ser feito somente entre os membros ativos, para que dessa forma não se permitisse que membros inativos tivessem uma parte do segredo. A mesma alteração foi feita para a autenticação dos membros do grupo, onde ao invés de fazer uma busca no smart card por um membro que pertença ao grupo, faria-se uma busca por um membro que estivesse ativo, removendo assim todos os privilégios dos membros inativos do grupo.

6 CONSIDERAÇÕES FINAIS

A participação dos autores no processo de desenvolvimento do software SGC do Projeto João de Barro culminou na implementação de diversas melhorias no software, de forma a garantir que ele se encontre em um nível de amadurecimento adequado às necessidades das Autoridades Certificadoras da ICP-Brasil, onde hoje já é utilizado com sucesso.

Entre os resultados deste trabalho estão melhorias significativas e necessárias no software, e no próprio processo de desenvolvimento e documentação do mesmo.

Dentre elas destaca-se a ramificação do software no projeto Ywyra, que também já é utilizado com sucesso em autoridades certificadoras intermediárias da ICP-Brasil. O aprendizado com o processo foi de grande valia, e o uso mais intenso do software e o retorno recebido dos usuários possibilitou a proposição e implantação de ainda mais melhorias, tanto no Ywapa quanto no Ywyra, bem como o vislumbramento de possibilidades futuras de novas implementações que serão descritas a seguir nas proposições de trabalhos futuros.

Outra melhoria significativa foi a implantação de novos algoritmos SHA-2 e ECDSA. Mais do que uma melhoria, isto é um requisito importante que torna o SGC pronto para atender as novas regulamentações da ICP-Brasil.

Destaca-se ainda a inclusão de mecanismos para garantir a compatibilidade entre bases de dados e backups antigos com versões mais recentes do sistema. Esta funcionalidade é indispensável uma vez que o software está sendo utilizado em diversas Autoridades Certificadoras mas continua em processo de evolução.

Além das atividades aqui documentadas, os autores ainda participaram da elaboração e ministração de treinamento para usuários e desenvolvedores do projeto, no próprio LabSEC e para o ITI.

6.1 TRABALHOS FUTUROS

Além de continuas melhorias que podem ser produzidas como fruto de respostas do uso do software em ambiente de produção, propõe-se a implementação do suporte a autoridades certificadoras de último nível, ou seja, aquelas que emitem certificados para entidades finais. Isto implicaria na implementação de um módulo para comunicação automatizada e segura com Autoridades de Registro (ARs) para os procedimentos de solicitação e revogação de certificados, bem como processos automatizados para emissão e publicação de LCR e suporte a repositórios LDAP. Sugere-se que esta implementação mantenha o critério de independência e interoperabilidade, através do uso de protocolos padronizados e abertos de comunicação entre entidades de uma ICP.

Outro trabalho futuro proposto, decorrente do primeiro acima citado, é a implantação de um software para Autoridades de Registro. Este, seguindo os mesmos padrões abertos, deve ser totalmente independente mas interoperável com o software de Autoridade Certificadora. O Software de Autoridade de Registro é também a interface com o usuário final, solicitante de certificados, através de uma interface pública.

Propõe-se ainda a implantação de outros padrões de interoperabilidade no software, como por exemplo suporte a outros Sistemas de Banco de Dados, e também suporte a comunicação com MSC através de PKCS#11. Tais melhorias permitiram o flexibilização e aumentariam ainda mais a interoperabilidade do software.

REFERÊNCIAS

[1] BRASIL. *Medida Provisória n° 2.200-2*. 2001.

[2] ITI. *Programa João de Barro*. jun. 2007. Disponível em: <http://www.iti.gov.br/twiki/bin/view/Swlivre/JoaoDeBarro>.

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

[4] FERGUSON, N.; SCHNEIER, B. *Practical Cryptography*. 1. ed. [S.l.]: Wiley Publishing, 2003.

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

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

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

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

[9] RIVEST, R.; SHAMIR, A.; ADLEMAN, L. *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. set. 1997. Disponível em: <http://theory.lcs.mit.edu/~rivest/rsapaper.pdf>.

[10] BARKER, E.; ROGINSKY, A. *Recommendation for the Transitioning of Cryptographic Algorithms and Key Sizes*. 2010. Disponível em: <http://csrc.nist.gov/publications/drafts/800-131/draft-800-131_transition-paper.pdf>.

[11] NIST. *FIPS 186 - Digital Signature Standard (DSS)*. maio 1994. Disponível em: <http://www.itl.nist.gov/fipspubs/fip186.htm>.

[12] NIST. *FIPS 186-3 - Digital Signature Standard (DSS)*. jun. 2009. Disponível em: <http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf>.

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

[14] HOUSLEY, R. et al. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. maio 2008. Disponível em: <http://www.ietf.org/rfc/rfc5280.txt>.

[15] CALLAS, J. et al. *OpenPGP Message Format*. nov. 2007. Disponível em: <http://tools.ietf.org/html/rfc4880>.

[16] ITI. *Glossário ICP-Brasil*. 2007. Disponível em: <http://www.iti.gov.br/twiki/pub/Certificacao/Legislacao/Glossario_ICP-Brasil-_Versao_1.3.pdf>.

[17] ADAMS, C. et al. *Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)*. set. 2005. Disponível em: <http://www.ietf.org/rfc/rfc4210.txt>.

[18] NYSTROM, M.; KALISKI, B. *PKCS #10: Certification Request Syntax Specification*. nov. 2000. Disponível em: <http://www.ietf.org/rfc/rfc2986.txt>.

[19] The OpenSSL Project. *OpenSSL: Documents, engine(3)*. mai 2010. Disponível em: <http://www.openssl.org/docs/crypto/engine.html>.

[20] Nokia Corporation. *Qt*. mai 2010. Disponível em: <http://qt.nokia.com/>.

[21] PostgreSQL Global Development Group. *PostgreSQL*. mai 2010. Disponível em: <http://www.postgresql.org/>.

[22] Red Hat, Inc. *Red Hat Enterprise Linux*. mai 2010. Disponível em: <http://www.redhat.com/rhel/>.

[23] The OpenSSL Project. *OpenSSL: The Open Source toolkit for SSL/TLS*. mai 2010. Disponível em: <http://www.openssl.org/>.

[24] CollabNet, Inc. *Subversion*. mai 2010. Disponível em: <http://subversion.apache.org/>.

[25] Edgewall Software. *Trac*. mai 2010. Disponível em: <http://trac.edgewall.org/>.