UNICEUB – CENTRO UNIVERSITÁRIO DE BRASÍLIA
FAET - FACULDADE DE CIÊNCIAS EXATAS E TECNOLOGIA
CURSO DE ENGENHARIA DA COMPUTAÇÃO

UMA ABORDAGEM DE INFRA-ESTRUTURA DE CHAVES PÚBLICAS PARA AMBIENTES CORPORATIVOS

BRASÍLIA – DF
2004

UNICEUB – CENTRO UNIVERSITÁRIO DE BRASÍLIA
FAET - FACULDADE DE CIÊNCIAS EXATAS E TECNOLOGIA
CURSO DE ENGENHARIA DA COMPUTAÇÃO

UMA ABORDAGEM DE INFRA-ESTRUTURA DE CHAVES PÚBLICAS PARA AMBIENTES CORPORATIVOS

por

BRUNO DE MELO SILVA
9965620 – FAET – UNICEUB

Projeto Final de Graduação

Prof. MC.***Luiz Otavio Botelho Lento***
Orientador

Brasília/DF, junho de 2004.

Agradecimentos

Aos meus pais, Luiz Carlos da Silva e Jacira de Melo Silva, por sua compreensão, apoio fraterno e dedicação.

Ao meu filho Pedro de Matta e Melo Silva, pelos momentos de descontração e ensinamentos que só uma criança pode proporcionar.

À minha namorada Soraya Chaibub Araújo de Lemos, pela paciência e incentivo, além de todo o apoio nestes cinco longos anos.

Ao professor Luiz Otavio, por sua orientação e sábias sugestões que foram fundamentais para a elaboração do projeto.

Ao meu irmão Vitor de Melo Silva, que nos deixou muito cedo, mas de certa forma continua conosco nesta jornada.

A todos os meus colegas e professores que participaram dessa maratona acadêmica e que espero na minha formatura.

A todos os meus familiares que me apoiaram e ajudaram em alguns momentos difíceis durante minha formação.

Resumo

Este trabalho visa o estudo dos padrões e definições de uma Infra-estrutura de Chaves Públicas (ICP) no modelo hierárquico e o desenvolvimento de um projeto de uma Infra-estrutura de Chaves Públicas privada para ambientes corporativos, de forma a implantar uma política de segurança e prover um aplicativo com as funcionalidades necessárias para a utilização dessa infra-estrutura.

O projeto abrange a criação de uma Autoridade Certificadora interna a uma empresa e a implantação de um aplicativo para efetivar o gerenciamento de chaves públicas e privadas e utilização das mesmas para criptografia assimétrica dos dados.

Este trabalho irá prover integridade, autenticidade e não-repúdio às informações dentro de uma corporação, diminuindo os prejuízos relativos a algumas fraudes eletrônicas, aumentando a segurança na guarda e tráfego de informações e evitando o gasto de se contratar uma Autoridade Certificadora externa.

Palavras-chave: ICP, certificados, AC, confidencialidade, não-repúdio.

LISTA DE FIGURAS

LISTA DE TABELAS

Sumário

1 – Introdução

Com o espantoso crescimento da informática e, posteriormente, o surgimento da Internet, a troca de informações através dos meios digitais se expandiu rapidamente. Empresas passaram a depender exclusivamente dessa tecnologia, agilizando seus negócios e expandindo suas abrangências no mercado. A comunicação se tornou simples, eficiente e barata. O comércio eletrônico, dependente dessa comunicação, foi responsável por cerca de US$100 bilhões em vendas, durante 1999.

Com esse crescimento, surgiram os problemas e os riscos de segurança. Diversos tipos de fraudes foram desenvolvidos se aproveitando de falhas na segurança. O setor de tecnologia de segurança passou por um crescimento explosivo em um curto período de tempo, devido a essas falhas de segurança. Variando desde novos desenvolvimentos em hardware criptográficos até o uso de cartões em infra-estruturas de chaves públicas, essa indústria continua aumentando a sua abrangência.

As empresas começaram a se deparar com problemas como documentos internos fraudados ou mesmo alterados. Dados cruciais de empresas, funcionários e clientes precisavam de uma segurança confiável, onde a leitura ou alteração de tais dados fossem restritos.

Surgiu a necessidade de se enviar informações onde apenas o destinatário fosse capaz de ler. E, além disso, surgiu a questão de como se faria uma transação garantindo um dos lados, ou seja, como seria possível garantir que a empresa ou cliente que enviou aquele documento, era ele mesmo (Não-repúdio). A criptografia, inicialmente criada com finalidade governamental, foi utilizada para resolver esses problemas.

A criptografia simétrica surgiu garantindo o sigilo dos dados enviados. Somente a outra parte poderia ler o documento. Mas algumas dificuldades surgiram na utilização dessa técnica. A troca da chave utilizada na criptografia se mostrava insegura, já que a chave é a mesma para os dois lados de uma transação. E, além desse problema de distribuição das chaves, ainda existia o fato de que para cada participante do ciclo de informações deveria ser criada uma nova chave criptográfica para a troca de mensagens.

O conceito de criptografia simétrica também não poderia garantir a autenticidade do remetente de uma mensagem. Estes problemas só vieram a ser superados com um novo conceito de cifragem chamado de criptografia assimétrica.

Essa criptografia garante o sigilo do dado enviado, a autenticidade e, principalmente, garante a identidade do indivíduo participante do ciclo da informação. Apenas o par da chave criptográfica criada pode desfazer a cifragem da outra chave.

A criptografia de chaves assimétricas ainda precisava ser aprimorada. Um indivíduo mal intencionado poderia gerar um par de chaves e enviar a sua chave pública para outros, se passando por alguém que ele não é. Ou então poderia interceptar uma chave pública em trânsito e o lado que deveriam receber a chave pública do primeiro, receberia a chave pública do indivíduo que esta intermediando a comunicação. Era preciso que um terceiro participante, que fosse confiável pelos outros dois, fosse criado. Esse terceiro iria validar os outros dois, permitindo assim, a troca de informações segura.

A infra-estrutura de chave pública foi desenvolvida com essa base. Com diversos tipos e modos de aplicação, sua utilização cresce cada vez mais no mercado. Foram criadas as Autoridades Certificadoras (AC) para executarem essa tarefa de validação. Os certificados digitais surgiram guardando as informações dos indivíduos. Após a assinatura digital de uma AC confiável em um certificado digital, o receptor desse certificado poderia validar a confiabilidade deste, utilizando a chave pública da AC que o assinou. Lembre-se que as chaves públicas são expostas a qualquer um. Daí surgiu o seu nome, que caracteriza essa propriedade.

O sistema de assinatura digital, os certificados digitais e os outros participantes dessa infra-estrutura passam a ter abordagens legislativas. Atualmente, o detentor de uma chave privada é totalmente responsável por ela, podendo responder a processos jurídicos, caso seja feito mal uso da mesma.

Este trabalho visa realizar um estudo sobre infra-estrutura de chaves públicas em um ambiente corporativo, permitindo a troca segura e confiável de informações dentro de uma empresa. Qualquer corporação pode se tornar a sua Autoridade Certificadora, gerando certificados para seus funcionários e, se desejar, para seus clientes e parceiros comerciais, reduzindo o custo de se implementar uma estrutura de chaves publicas já preparada por uma AC comercial, como a VeriSign.

2 – Criptografia

A palavra criptografia tem origem grega (kriptos = escondido, oculto e grifo = grafia, escrita) e define a arte ou ciência de escrever em cifras ou em códigos, utilizando um conjunto de técnicas que torna uma mensagem incompreensível, chamada comumente de texto cifrado, através de um processo chamado cifragem, permitindo que apenas o destinatário desejado consiga decodificar e ler a mensagem com clareza, no processo inverso, a decifragem.

Criptografia é a ciência de escrever ocultamente e hoje, sem dúvida, é a maneira mais segura de se enviar informações através de um canal de comunicação inseguro como, por exemplo, a Internet.

A criptografia representa um conjunto de técnicas que são usadas para manter a informação segura. Estas técnicas consistem na utilização de chaves e algoritmos de criptografia. Tendo conhecimento da chave e do algoritmo usado é possível desembaralhar a mensagem recebida.[1]

Fica claro que o objetivo é transformar palavras escritas de forma a tornarem ilegíveis para receptores não autorizados. O destinatário autorizado pode transformar a palavra recebida novamente em mensagem compreensível. Por exemplo, esta é uma mensagem a ser cifrada:

"A mudança de regra está nos obrigando a fazer essa enriquecedora monografia"

Esta é a mensagem já cifrada:

"~çk&*(çd)(*&¨%$jrijgo~´oruTkh*,MjHbbvRyh565$#$$%¨7><><;;Ç^ç´{+--"."

2.1 - As raízes da criptografia

A idéia da criptografia tem milhares de anos: generais gregos e romanos usaram a criptografia para enviar mensagens em código aos comandantes de campo. Estes sistemas antigos eram baseados em duas técnicas: substituição e transposição.

A **SUBSTITUIÇÃO** baseia-se na troca de cada letra ou grupo de letras da mensagem de acordo com uma tabela de substituição. As substituições podem ser subdivididas em:

1. **Substituição simples ou monoalfabética**: é o tipo de cifra na qual cada letra da mensagem é substituída por outra, de acordo com uma tabela baseada geralmente num deslocamento da letra original dentro do alfabeto. A cifragem de (Lolius Caessar) Júlio César que não confiava em seus mensageiros, por exemplo, substituía a letra "A" por "D", "B" por "E" e assim por diante. Só aqueles que conheciam a regra "desloque 3 à frente" podia decifrar a mensagem original. Algumas cifras de substituição usam o mesmo esquema de substituição para todas as letras; outros usam esquemas diferentes para letras diferentes;
2. **Substituição monofônica**: funciona como a substituição monoalfabética, mas cada caracter da mensagem original pode ser mapeado para um ou vários caracteres na mensagem cifrada. Por exemplo, a letra A seria substituída pelos números 7, 34 ou 78.
3. **Substituição polialfabética**: consiste em utilizar várias cifras de substituição simples, em que as letras ou blocos da mensagem são substituídos por valores diferentes;
4. **Substituição de polígramos ou blocos**: utiliza um grupo de caracteres ao invés de um único caractere individual para a substituição da mensagem. Por exemplo, "ABA" pode corresponder a "MÃE" e "ABB" corresponder a "JKI";
5. **Substituição por deslocamento**: ao contrário da cifra de César, não usa um valor fixo para a substituição de todas as letras. Cada letra tem um valor associado para a rotação através de um critério. Por exemplo, cifrar a palavra "CARRO" utilizando o critério de rotação "023", seria substituir "C" pela letra que está 0(zero) posições à frente no alfabeto, o "A" pela letra que está 2 (duas) posições a frente, e assim por diante, repetindo-se o critério se necessário, gerando, assim, o criptograma "CCURQ".

A **TRANSPOSIÇÃO** baseia-se na mistura dos caracteres da mensagem. Temos como exemplo de um sistema de transposição em que a mensagem é escrita em uma tabela linha por linha e depois lida coluna por coluna.

No início do século XX, vários mecanismos eletromecânicos foram construídos na Europa e nos Estados Unidos com a finalidade de codificar mensagens enviadas por telégrafo ou por rádio. Estes sistemas baseavam-se principalmente na substituição, pois não havia uma forma específica para armazenar uma mensagem inteira usando técnicas de transposição. Hoje em dia, algoritmos de criptografia executados em computadores digitais de alta velocidade usam combinações de transposição e substituição, assim como outras funções matemáticas.[1]

2.2 - O objetivo da criptografia

O objetivo da criptografia é tornar impossível a recuperação de um texto em claro a partir de um texto cifrado sem a chave correspondente e, além disso, dificultar ao máximo a chance de que se descubra sem autorização qual a chave que tornaria isso possível. Muitos sistemas modernos de criptografia realizam essa tarefa com facilidade. De fato, existem atualmente algoritmos criptográficos sem falhas conhecidas disponíveis para serem utilizados.

Os sistemas modernos de criptografia consistem de dois processos complementares:

· **Cifragem**: Processo pelo qual uma mensagem (o texto em claro) é transformada em uma segunda mensagem (o texto cifrado) usando uma função complexa (o algoritmo de criptografia) e uma chave criptográfica especial;

· **Decifragem**: O processo inverso, pelo qual o texto cifrado é transformado no texto em claro usando uma segunda função complexa e uma chave de decifragem. Em alguns sistemas criptográficos a chave criptográfica e a chave de decifragem são iguais, em outros, diferentes.

A tendência atual é utilizar algoritmos conhecidos e largamente testados, cuja eficácia e robustez sejam comprovadas, sendo que a segurança reside totalmente na chave secreta, que é o parâmetro da função. Esta chave numérica deve ter tamanho suficiente para evitar sua descoberta por teste exaustivo pois, como o algoritmo é conhecido, pode ser utilizado para combinar todas os possíveis valores da chave secreta. As chaves devem ter, portanto, um tamanho que inviabilize este tipo de ataque, exigindo do atacante um tempo de processamento excessivamente longo, mesmo utilizando capacidade computacional elevada.

Os algoritmos criptográficos são classificados em simétricos e assimétricos, sendo que cada esquema tem vantagens e desvantagens, e a abordagem utilizada normalmente é combiná-los de modo adequado, para eliminar as desvantagens de cada um e aproveitar suas vantagens comparativas.

2.3 - Algoritmos simétricos

Estes são os algoritmos convencionais de criptografia, onde a mesma chave secreta é utilizada tanto para cifrar como para decifrar uma mensagem, devendo ser conhecida por ambos os lados do processo. Este é o grande problema do método, pois a chave tem de ser entregue aos participantes de modo seguro, e as transações só podem ser realizadas depois disso.

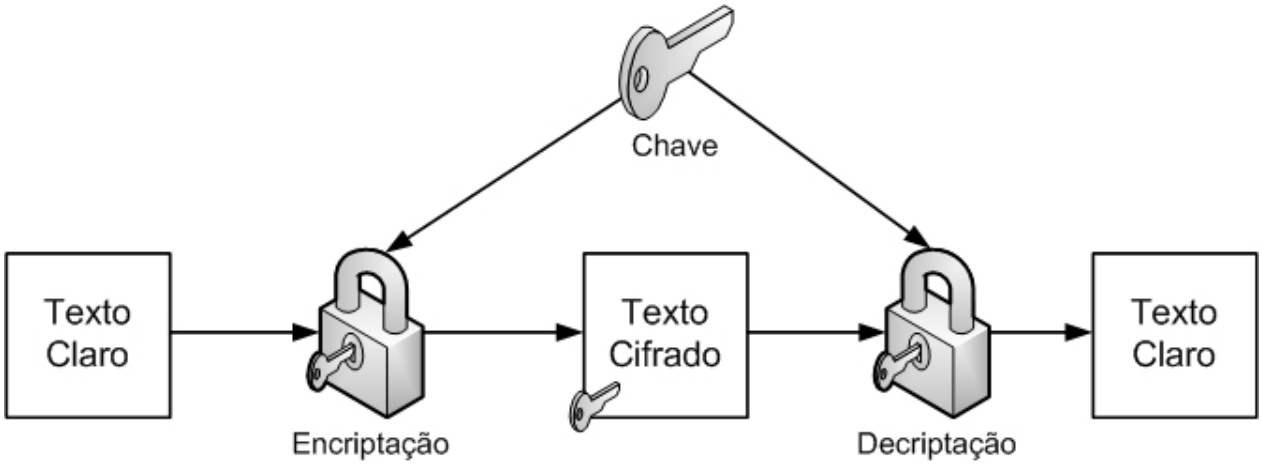

Figura 1 - Criptografia Simétrica

O fato de ambos os lados conhecerem a chave também leva à possibilidade de repúdio da transação, pois um lado pode sempre alegar que o outro usou a chave e realizou a transação em seu nome, indevidamente.

Como cada par de participantes deve ter uma chave própria, o número de chaves necessárias para comunicação segura entre muitos participantes cresce combinatoriamente, com agravante adicional de que todas essas chaves são secretas e devem ser protegidas adequadamente. Ou seja, um participante do ciclo de criptografia deverá ter a chave de todos os outros para se comunicar com cada um deles. Isso inviabiliza o uso destes algoritmos isoladamente em certas aplicações.

Os algoritmos de chave simétrica são usados para cifrar a maioria dos dados ou fluxos de dados. Estes algoritmos são projetados para serem bem rápidos e (geralmente) terem um grande número de chaves possíveis. Os melhores algoritmos de chave simétrica oferecem boa segurança quando os dados são cifrados com determinada chave, e dificilmente pode-se decifrar os dados sem possuir a mesma chave. Como a criptografia é sempre uma carga

adicional ao processamento, esta vantagem é importante e deverá ser utilizada adequadamente.

Os algoritmos de chave simétrica podem ser divididos em duas categorias: de bloco e de fluxo.

Algoritmos de bloco: Cifram os dados a partir de blocos, ou seja, se o dado a ser cifrado é um texto, esse texto será dividido em blocos e a criptografia será aplicada em cima de cada bloco. Um problema com essa cifragem é que se o mesmo bloco de texto simples aparecer em dois lugares, ele encriptará o mesmo texto, gerando assim, um padrão de repetição.

Algoritmos de fluxo: Cifram os dados byte a byte. O dado a ser criptografado não é cifrado por blocos, como o anterior e sim, serialmente. A informação vai sendo criptografada do inicio ao fim, sem separações.[2]

Há muitos algoritmos de chave simétrica em uso atualmente. Alguns dos algoritmos mais comuns no campo da segurança são:

DES - o Padrão para Criptografia de Dados (Data Encryption Standard) foi adotado como padrão pelo governo dos EUA em 1977, e como padrão ANSI em 1981. O DES é um algoritmo de bloco que usa uma chave de 56 bits e tem diferentes modos de operação, dependendo da finalidade com que é usado. O DES é um algoritmo poderoso, mas o seu reinado no mercado começou a ruir em janeiro/1997, quando a empresa RSA Data Security Inc. (que detém a patente do sistema criptográfico RSA) - decidiu colocar o DES à prova, oferecendo um prêmio de US$ 10 mil à primeira pessoa ou instituição que decifrasse uma frase criptografada com o DES (vencido o primeiro desafio, a RSADSI decidiu repeti-lo a cada semestre, condicionando o pagamento do prêmio à quebra do recorde de tempo estabelecido até o momento). Atualmente uma máquina preparada para a tarefa é capaz de decifrar uma mensagem cifrada com o DES em poucas horas.

DESX - é uma simples modificação do algoritmo DES, constituída em duas etapas. Este método parece melhorar a segurança do algoritmo, dificultando a busca da chave.

Triple-DES - é uma maneira de tornar o DES pelo menos duas vezes mais seguro, usando o algoritmo de criptografia três vezes, com três chaves diferentes. Usar o DES duas vezes com duas chaves diferentes não aumenta tanto a segurança quanto se poderia pensar devido a um

tipo teórico de ataque conhecido como meet-in-the-midle (encontro no meio), com o qual o atacante tenta cifrar o texto limpo simultaneamente com uma operação do DES e decifrar o texto com outra operação, até que haja um encontro no meio. Atualmente, o Triple-DES está sendo usado por instituições financeiras com uma alternativa para o DES.

IDEA - o International Data Encryption Algorithm (IDEA - Algoritmo de Criptografia de Dados Internacional) foi desenvolvido em Zurique, na Suíça, por James L. Massey e Xuenjia Lai, e publicado em 1990. O IDEA usa chave de 128 bits, e é bastante poderoso.

RC2 - este algoritmo de bloco foi desenvolvido originalmente por Ronald Rivest, e mantido em segredo pela RSA Data Security. Foi revelado por uma mensagem anônima na Usenet em 1996, e parece ser relativamente poderoso (embora algumas chaves sejam vulneráveis). O RC2 é vendido com uma implementação que permite a utilização de chaves de 1 a 2048 bits.

RC4 - Inventado em 1987 pela RSA, nunca teve o seu algoritmo de funcionamento interno publicado. Esse segredo possuía interesses financeiros e não de segurança. A empresa esperava que mantendo-o em segredo, ninguém mais o implementaria e o comercializaria. É uma cifragem muito utilizada hoje em dia, até fazendo parte no protocolo de comunicacao SSL (Security Socket Layer).

RC5 - este algoritmo de bloco foi desenvolvido por Ronald Rivest e publicado em 1994. O RC5 permite que o tamanho da chave, o tamanho dos blocos de dados e o número de vezes que a criptografia será realizada seja definida pelo usuário.

Blowfish - é um algoritmo de criptografia em bloco, rápido, compacto e simples, inventado por Bruce Schneier. O algoritmo permite a utilização de uma chave de tamanho variável, de até 448 bits, e é otimizado para executar em processadores de 32 ou 64 bits. Não é patenteado e foi colocado em domínio público.[3]

2.4 - Algoritmos assimétricos

A existência da criptografia de chave pública foi postulada pela primeira vez em meados de 1975 por Withfield Diffie e Martin Hellman. Os dois pesquisadores, na época na universidade de Stanford, escreveram um artigo em que pressuponham a existência de uma técnica criptográfica com a qual a informação criptografada com uma chave poderia ser decifrada por uma segunda chave, aparentemente sem relação com a primeira. Robert Merkle, então estudante em Berkeley que tinha idéias semelhantes mas, devido à lentidão do processo de publicação acadêmica, seus artigos só foram publicados quando a idéia de criptografia de chave pública já era bem conhecida.

Desde então, vários sistemas de chave pública foram desenvolvidos. Infelizmente, houve muito menos desenvolvimento nos algoritmos de chave pública em relação aos de chave simétrica. A razão disso tem a ver com o modo como esses algoritmos são criados. Um bom algoritmo de chave simétrica simplesmente embaralha os dados de entrada, dependendo da chave utilizada. O desenvolvimento de um novo algoritmo de chave simétrica depende da criação de novas maneiras de executar esta operação de forma confiável. Os algoritmos de chave pública são baseados na teoria dos números. Especialistas dizem que "O desenvolvimento de novos algoritmos de chave pública requer a identificação de novos problemas matemáticos com propriedades particulares".

Os algoritmos assimétricos utilizam-se de duas chaves diferentes, uma em cada extremidade do processo. As duas chaves são associadas através de um relacionamento matemático, pertencendo a apenas um participante, que as utilizará para se comunicar com todos os outros de modo seguro.

Essas duas chaves são geradas de tal maneira que a partir de uma delas não é possível calcular a outra a um custo computacional viável, possibilitando a divulgação de uma delas, denominada chave pública, sem colocar em risco o segredo da outra, denominada chave secreta ou privada.

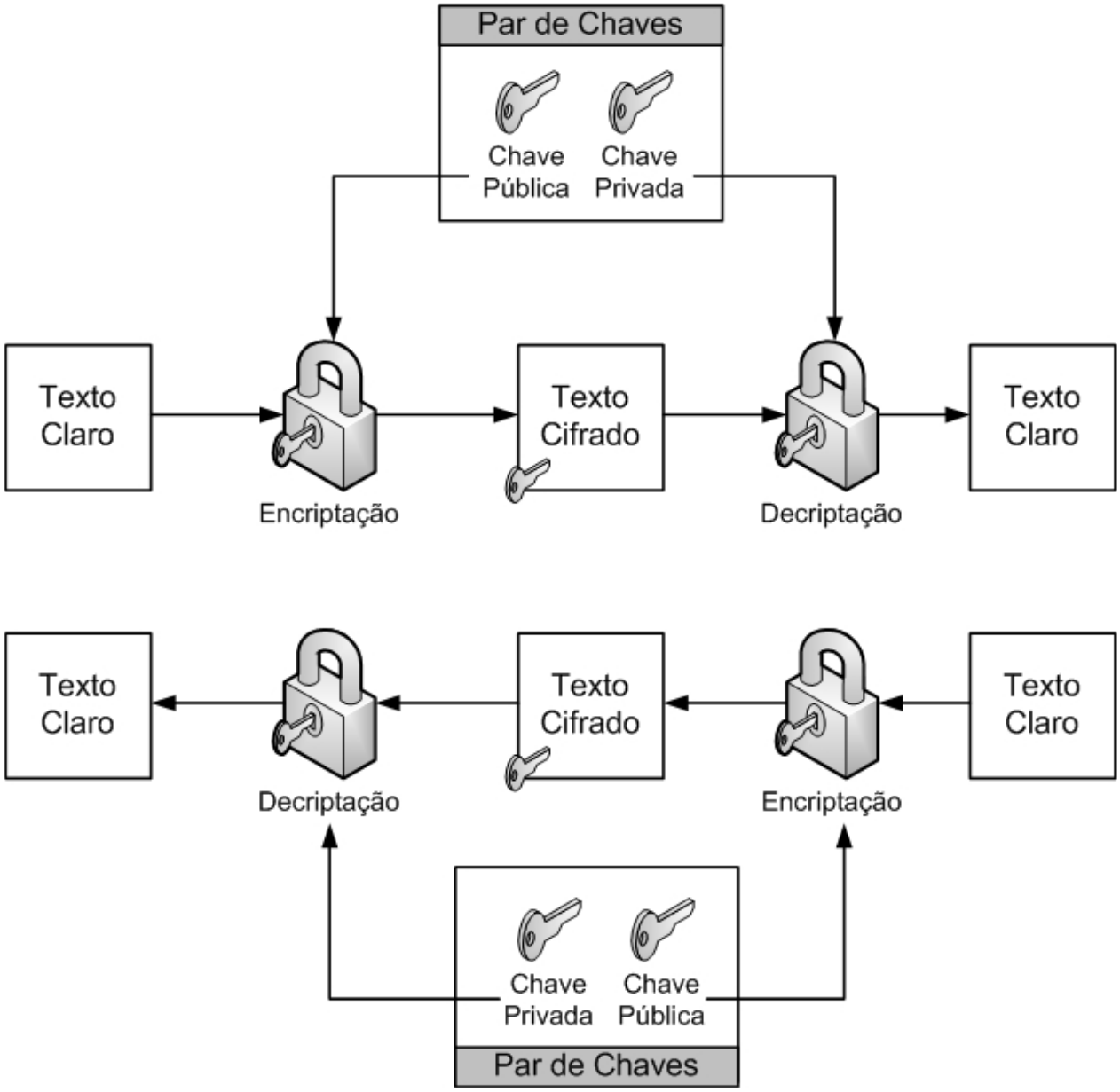

Figura 2 – Criptografia Assimétrica

Os principais sistemas de chaves públicas atualmente em uso são:

Diffie-Hellman - Um sistema para troca de chaves criptográficas entre partes. Na verdade, não é um método de criptografia ou decifragem, é um método para troca de chave secreta compartilhada por meio de um canal de comunicação público. Com efeito, as duas partes estabelecem certos valores numéricos comuns e cada uma delas cria uma chave. As transformações matemáticas das chaves são intercambiadas. Cada parte calcula então uma terceira chave (a chave de sessão) que não pode ser descoberta facilmente por um atacante que conheça os valores intercambiados.

ElGamal - Batizado com o nome de seu criador, Taher ElGamal, é um sistema criptográfico de chave pública baseado no protocolo de troca de chaves de Diffie- Hellman. O ElGamal pode ser utilizado para criptografia e assinatura digital, de forma semelhante ao algoritmo RSA.

DSS - O Digital Signature Standard (DSS - Padrão de Assinatura Digital) foi desenvolvido pela Agência Nacional de Segurança (NSA), e adotado como Padrão Federal de Processamento de Informação (FIPS) pelo Instituto Nacional de Padrões Tecnologia (NIST) dos EUA. O DSS é baseado no Algoritmo de Assinatura Digital - DSA (Digital Signature Algorithm) - que permite a utilização de qualquer tamanho de chave, embora no DSS FIPS só sejam permitidas chaves entre 512 e 1024 bits. O DSS só pode ser usado para a realização de assinaturas digitais, embora haja implementações do DSA para criptografia.

RSA - RSA é um sistema criptográfico de chave pública conhecido, desenvolvido por Ronald Rivest, Adi Shamir e Leonard Adleman, então professores do MIT (Instituto de Tecnologia de Massachusets). O RSA utiliza criptografia em blocos e possui uma segurança muito forte, devido ao alto poder computacional necessário para se tentar quebrar uma chave RSA. Pode tanto ser usado para cifrar informações como para servir de base para um sistema de assinatura digital. As assinaturas digitais podem ser usadas para provar a autenticidade de informações digitais. A chave pode ser de qualquer tamanho, dependendo da implementação utilizada.[4] [14]

2.5 - Assinatura digital

Outra grande vantagem dos algoritmos assimétricos, particularmente o RSA, que é o mais conhecido e utilizado atualmente, é que o processo funciona também na criptografia no outro sentido, da chave secreta para a chave pública, o que possibilita implementar o que se denomina assinatura digital.

O conceito de assinatura é o de um processo que apenas o signatário possa realizar, garantindo dessa maneira sua participação pessoal no processo. Como a chave secreta é de posse e uso exclusivo de seu detentor, um processo de cifragem usando a chave privada do signatário se encaixa nesse conceito, permitindo, assim, a geração de uma assinatura por um processo digital.

No caso da assinatura digital, é inadequado cifrar toda a mensagem ou documento a ser assinado digitalmente devido ao tempo gasto na criptografia de um documento utilizando chaves assimétricas. A criptografia é aplicada apenas sobre um identificador unívoco do mesmo. Normalmente é utilizado como identificador o resultado da aplicação de uma função tipo HASH, que mapeia um documento digital de tamanho qualquer num conjunto de bits de tamanho fixo. Ao valor do HASH podem ainda ser anexados a data/hora, número de seqüência e outros dados identificadores, e este conjunto é então cifrado com a chave secreta do signatário constituindo a assinatura digital do documento. A função de HASH será explicada em seguida.

Qualquer participante pode verificar a autenticidade de uma assinatura digital, bastando decifrá-la com a chave pública do signatário, o qual todos podem ter acesso. Se o resultado é significativo, está garantido o uso da chave secreta correspondente na assinatura, e portanto sua autenticidade. Resta ainda comprovar a associação da assinatura ao documento, o que é feito recalculando o HASH do documento recebido e comparando-o com o valor incluído na assinatura. Se forem iguais, prova-se ainda a ligação com o documento, assim como a integridade (não alteração) do mesmo. Uma vez que a verificação é realizada utilizando a chave pública, sua validação pode ser realizada por terceiros, tais como árbitros e auditores.

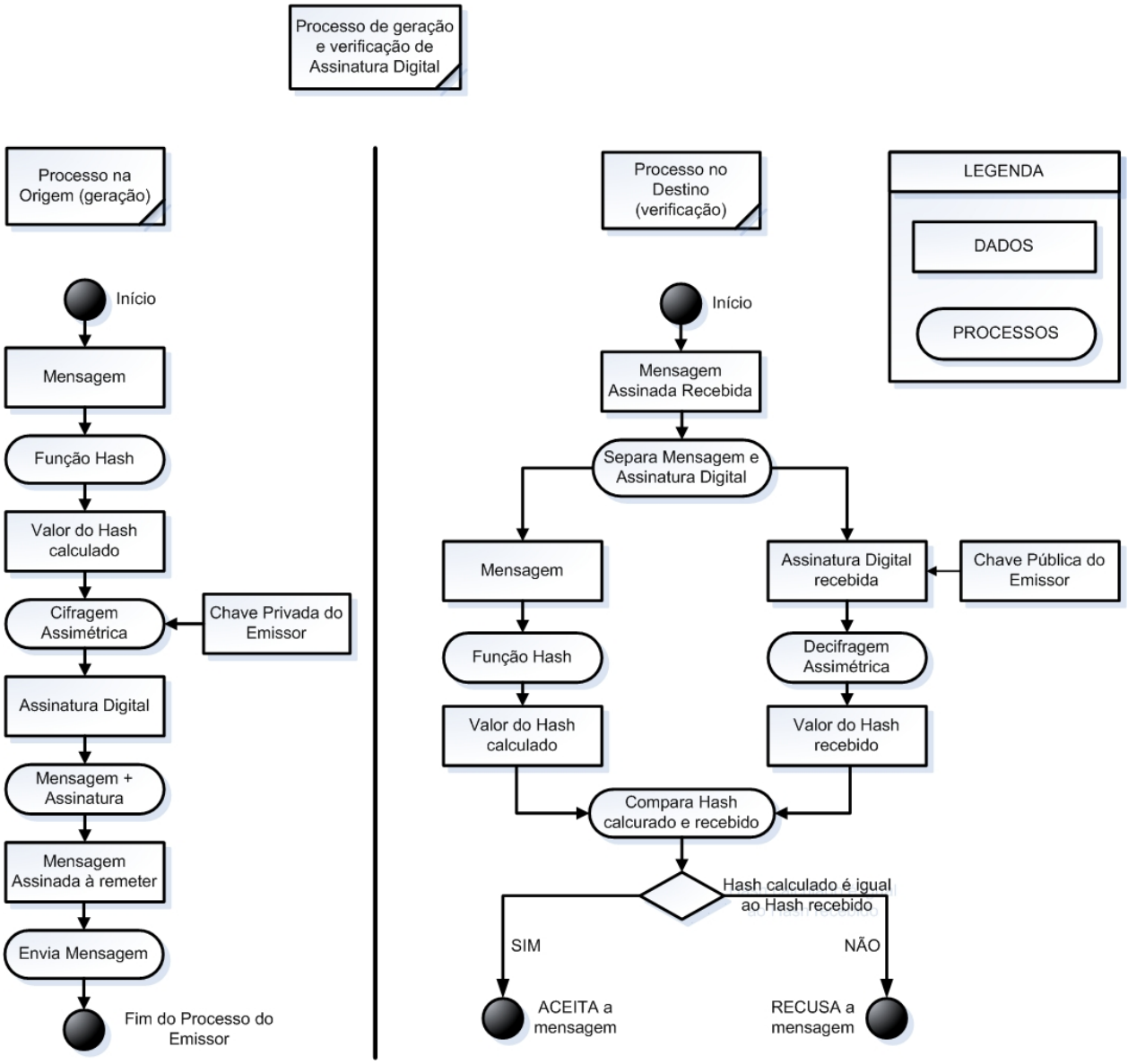

Figura 3 - Processo de Verificação de Assinatura Digital.

2.5.1 - HASH

A função HASH define uma criptografia de modo unidirecional, ou seja, efetua uma criptografia que não apresenta volta. Uma vez utilizada uma função de HASH em um documento, deve ser computacionalmente inviável obter o documento digital original a partir do valor resultante da função HASH sobre ele. Um fator interessante é que documentos de tamanhos e formatos diferentes podem gerar um mesmo valor de HASH. Mas nunca podem ser obtidos a partir destes valores.

Os algoritmos conhecidos como HASH são utilizados para codificação de mensagens. Os principais são:

HMAC - o Hashed Message Authentication Code (Código de Autenticação de Mensagens por Confusão), é uma técnica que usa uma chave secreta e uma função de codificação para criar um código secreto de autenticação de mensagem. O método HMAC reforça uma função de codificação existente de forma a torná-la resistente a ataques externos, mesmo que a própria função de codificação esteja de certa forma comprometida.

MD2 - Message Digest #2 (Codificação de Mensagem #2), desenvolvido por Ronald Rivest. Esta é a mais segura das funções de codificação de mensagem de Rivest, mas demora mais para calcular e produz algumas colisões em seu cálculo. Produz uma codificação de 128 bits.

MD4 - Message Digest #4 (Codificação de Mensagem #4), também desenvolvido por Ronald Rivest. Este algoritmo de codificação de mensagem foi desenvolvido como uma alternativa rápida para o MD2. Depois, o MD4 mostrou ser pouco seguro. Ou seja, é possível encontrar dois arquivos que produzem o mesmo código MD4 sem uma pesquisa de força bruta. O MD4 produz uma codificação de 128 bits.

MD5 - Message Digest #5 (Codificação de Mensagem #5), também desenvolvido por Ronald Rivest. O MD5 é uma modificação do MD4 que inclui técnicas para torná-lo mais seguro e mais rápido. Embora largamente usado, foram descobertas algumas falhas nele em meados de 1996, que permitiam calcular alguns tipos de colisão. Como resultado, sua popularidade vem caindo. O MD5 produz uma codificação de 128 bits.

SHA1 - O Secure Hash Algorithm se parece muito com o MD5. O SHA1 incorpora pequenas mudanças, como produzir uma codificação maior com 160 bits. Os cálculos internos são mais fortes do que o do MD5. Variantes do SHA1 que produzem variantes de 192 a 256 bits estão em desenvolvimento.[5]

2.6 - Confidencialidade, autenticidade, não-repúdio

2.6.1 – Confidencialidade

Vamos supor que Antônio queira enviar, através de um canal de comunicação inseguro (a Internet, por exemplo) uma mensagem sigilosa M a Benedito, utilizando um sistema criptográfico de chave pública. Após obter PB, a chave pública de Benedito (essa chave deve ser conseguida de forma segura, como será apresentado no projeto), ele cifra a mensagem M, obtendo a mensagem cifrada PB (M). Benedito, após receber a mensagem cifrada, aplica a ela sua chave secreta SB, obtendo a mensagem inicial M. Observe que apenas Benedito conseguirá ler a mensagem, já que é inviável a qualquer outra pessoa, somente de posse da chave pública de Benedito (PB), deduzir sua chave secreta (SB).

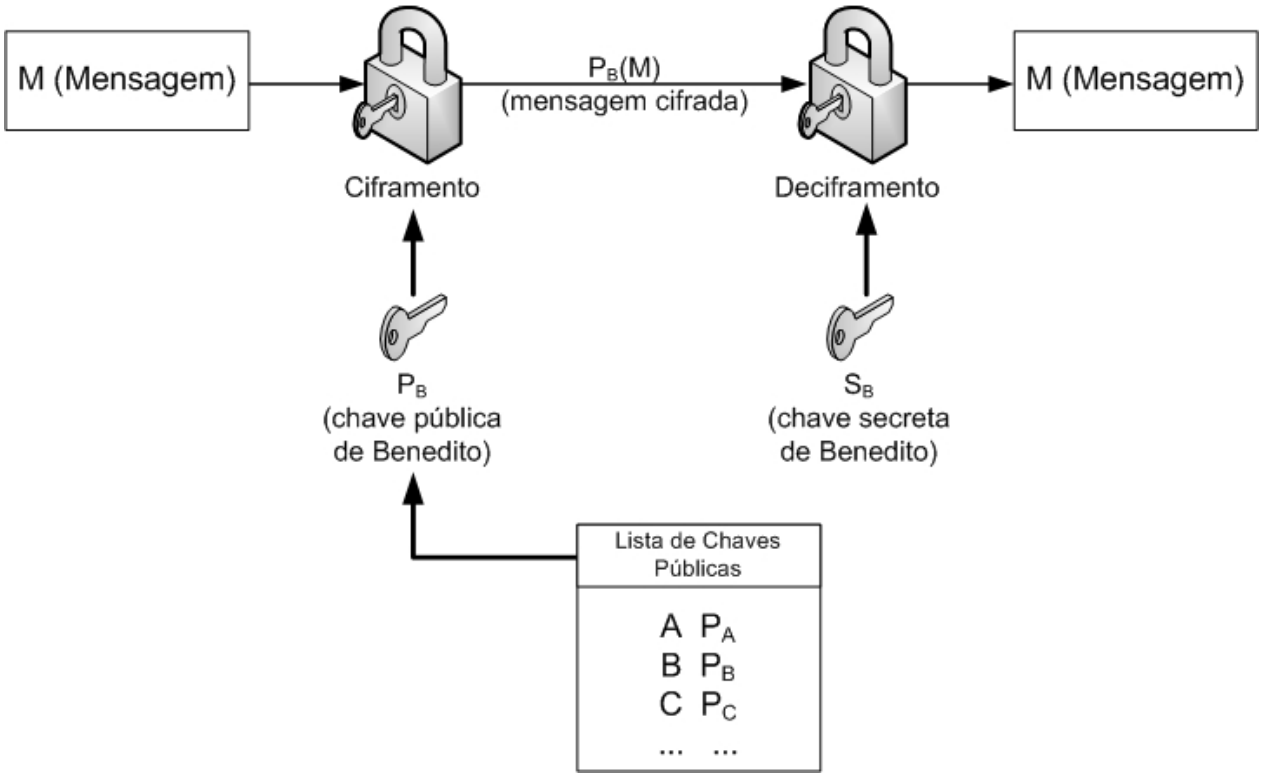

Figura 4 – Confidencialidade dos dados

2.6.2 - Não-repúdio

No exemplo anterior, Benedito não tem como afirmar, com certeza, se a mensagem foi efetivamente enviada por Antônio ou se por algum impostor. Vamos supor, agora, que Antônio queira enviar uma mensagem M não sigilosa, mas que, por outro lado, queira assinar tal mensagem, de modo a permitir que Benedito, ou qualquer outra pessoa que tenha acesso à mensagem, possa ter certeza, ao examiná-la, que seu remetente é realmente Antônio. Antônio codifica a mensagem M com sua chave secreta SA, obtendo SA (M). Benedito, ou qualquer outra pessoa, pode confirmar que a mensagem provém realmente de Antônio, através da aplicação da chave pública de Antônio (PA) à mensagem SA (M). Note que as chaves se completam, tornando possível o caminho de ida e volta utilizando as chaves pública e privada.

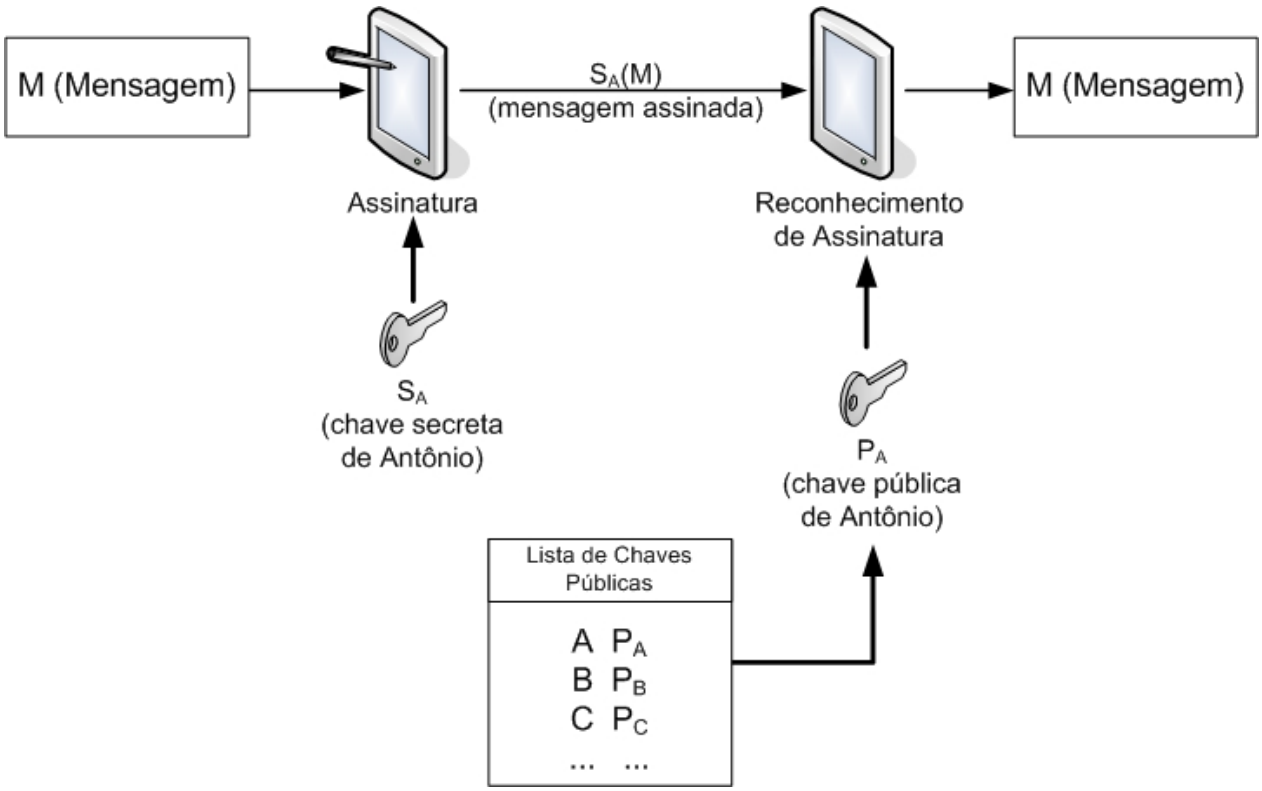

Figura 5 - Não-repúdio

2.6.3 - Confidencialidade, Autenticidade e Não-Repúdio

Neste terceiro exemplo, suponha que Antônio queira enviar a Benedito, novamente através de um canal de comunicação inseguro, uma mensagem sigilosa M, permitindo que Benedito possa se certificar da procedência da mensagem e de que essa mensagem não foi alterada. Inicialmente, Antônio assina a mensagem M, utilizando sua chave secreta SA e, em seguida, cifra a mensagem resultante, utilizando a chave pública de Benedito (PB), obtendo, assim, PB (SA (M)). Benedito, ao receber a mensagem cifrada e assinada por Antônio, aplica, em primeiro lugar, sua chave secreta (SB), obtendo, dessa forma, a mensagem criptografada pela chave secreta de Antônio. Em seguida, Benedito, utilizando a chave pública de Antônio (PA), recupera a mensagem original, M. Dessa forma pode-se concluir que a mensagem enviada foi confidencial, autentica e existe a certeza do remetente da mensagem.

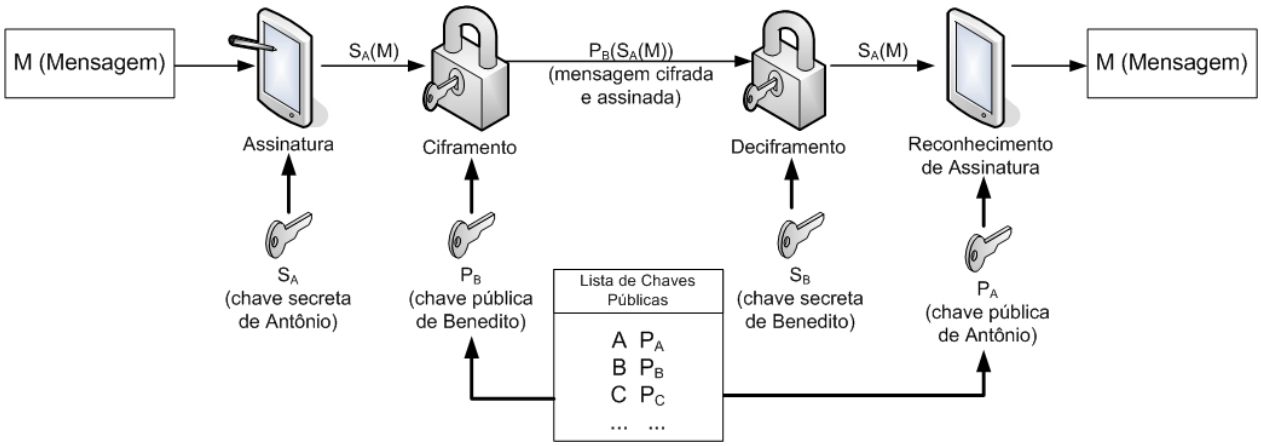

Figura 6 - Confidencialidade, Autenticidade e Não-Repúdio

3 – Infra-estrutura de Chaves Públicas

Uma Infra-estrutura de Chaves Públicas (ICP) é um sistema de segurança baseado em tecnologia para estabelecer e garantir a confiabilidade de chaves públicas de criptografia. A criptografia de chaves públicas tem se apresentado como um importante mecanismo de segurança para o fornecimento de serviços de autenticação, geração de provas, integridade de dados e confidencialidade para operações internas e externas de e-business. Quando implementada como um componente integral de uma solução de habilitação de confiabilidade, uma ICP pode contribuir para a otimização da velocidade e do valor das transações de e-business.

A infra-estrutura de chaves públicas atrela as chaves públicas às suas entidades, possibilitando que outras entidades verifiquem a validade das chaves públicas e disponibilize os serviços necessários para o gerenciamento das chaves que trafegam em um sistema distribuído.

O objetivo maior dessa arquitetura de segurança moderna é proteger e distribuir a informação que é necessária em ambientes altamente distribuídos, nos quais os usuários, recursos e empresas podem estar em lugares diferentes.

Em transações comerciais convencionais, por exemplo, os clientes e os comerciantes baseiam-se em cartões de créditos (p.ex., VISA ou Mastercard) para completar os aspectos financeiros das transações. O vendedor autentica o cliente através da comparação de assinaturas ou verificando um documento de identidade, como um RG. O vendedor se baseia na informação contida no cartão de crédito e na informação obtida junto ao emissor do cartão de crédito para garantir que o pagamento será recebido. Da mesma forma, o cliente faz a transação sabendo que ele pode rejeitar a conta se o vendedor falhar no fornecimento do bem ou serviço. O emissor do cartão de crédito é o terceiro confiável nesse tipo de transação.

O mesmo modelo pode ser aplicado em uma transferência de informação (como um cadastro de pessoa física), mesmo sabendo que o consumidor e o vendedor talvez nunca se encontrem. O vendedor não pode comparar as assinaturas ou pedir um documento de identidade. Eles podem estar separados por centenas de quilômetros. Mas ambos precisam ser capazes de assegurar que a outra parte é legítima para que haja a troca de informações.

A infra-estrutura de chave pública é uma combinação de software, tecnologias de encriptação e serviços que permite às empresas obterem segurança em suas comunicações e transações comerciais via rede, integrando certificados digitais, criptografia de chave pública

e autoridades certificadoras numa arquitetura de segurança de redes completa para, dessa forma, criar uma estrutura de confiança para os dois lados da transação.

A ICP consegue assegurar confidencialidade, integridade e não-repúdio de uma maneira difícil de ser fraudada e que se apresenta de forma transparente para o usuário. Estes dois pontos, transparência aliada à forte base técnica de seus mecanismos, denotam o aspecto forte desta tecnologia.[2]

3.1 – A arquitetura de uma ICP

Existem, atualmente, dois tipos básicos de arquiteturas de ICP criadas para solucionar o problema de validação dos participantes em um processo de troca de informações: A arquitetura Hierárquica e a Mista. O projeto em questão abrange a solução Hierárquica, que é a mesma utilizada pela ICP-Brasil (Infra-estrutura de Chaves Públicas-Brasil).

3.1.1 - Arquitetura Hierárquica

A arquitetura tradicional utilizada pela ICP é a Hierárquica. Nesta arquitetura, múltiplas AC's prestam serviços de infra-estrutura e as Autoridades Certificadoras possuem uma relação de confiança em escala Hierárquica. Nessa arquitetura, todas as AC's confiam em uma AC central chamada de Raiz. Com exceção da AC Raiz, todas as outras AC's possuem uma única AC superior.
O modelo abaixo representa essa arquitetura.

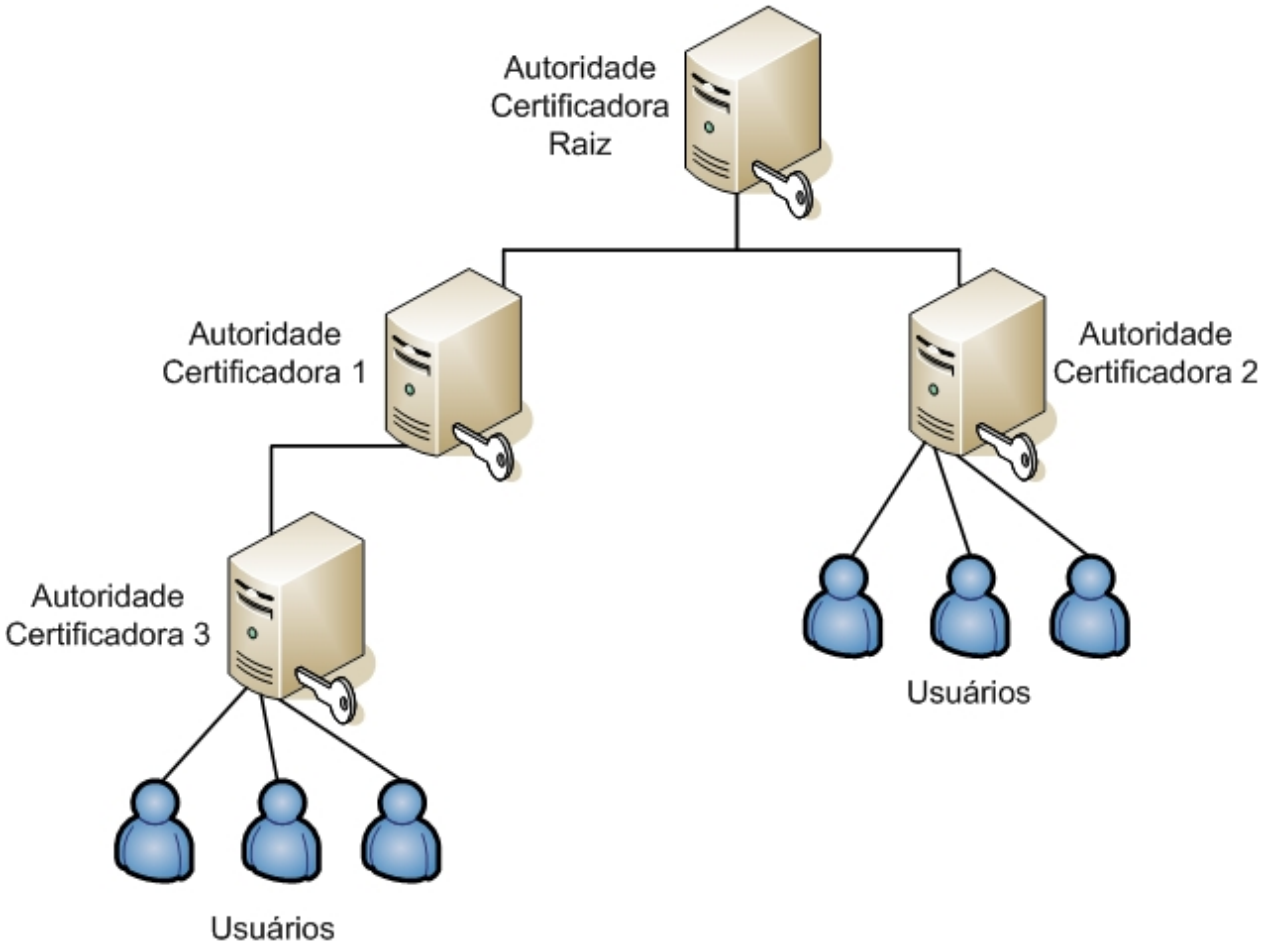

Figura 7 - Arquitetura Hierárquica

A AC Raiz é o topo da hierarquia, ou seja, o certificado digital dessa AC é um certificado auto-assinado. Ninguém assina seu certificado, pois ela não é dependente de nenhuma outra AC. Isso é necessário para garantir a validação da hierarquia das outras AC's. A partir daí, cada AC que for criada nessa estrutura terá seu certificado assinado pela AC Raiz. Seguindo a lógica da figura, temos o seguinte esquema:

A AC Raiz assina o seu próprio certificado e se prepara para receber mais AC's em sua hierarquia. A AC 1 e a AC 2 decidem entrar na arquitetura oferecida pela AC Raiz. Elas enviam uma REQUISIÇÃO DE ASSINATURA DE CERTIFICADO (CSR) para a AC Raiz e recebem desta seus certificados já assinados pela chave secreta da Raiz. A AC 2 decide que irá gerar certificados digitais diretamente aos seus clientes. Então, os clientes geram um CSR (Através de um browser ou um aplicativo disponibilizado pela AC 2) e, a partir desse CSR, a AC 2 devolve um certificado assinado por ela.

No outro caso, a AC 1 não assina certificados de usuários em geral, apenas de empresas ou outras AC's. A AC 3 envia um CSR para a AC 1 que assina e devolve um certificado assinado por ela para a AC 3. A AC 3, por sua vez, recebe CSR's de clientes.
Qual a importância dessa cadeia de assinaturas?

Usaremos um exemplo de uma empresa que vende automóveis pela Internet e, para isso, exige que o cliente apresente seu Certificado Digital.

O cliente acessa o site da empresa e compra um automóvel, utilizando o seu certificado digital. O próprio site da empresa captura o certificado digital do cliente (lembre-se que o certificado digital é PÚBLICO) e irá tentar conferir a validade deste. Essa empresa precisa ter, em primeiro lugar, o certificado digital da AC Raiz. Esse certificado é de extrema importância e precisa ser confiável, podendo ser conseguido junto à própria AC Raiz. Aproveitando a figura da arquitetura hierárquica acima, digamos que o cliente que comprou o automóvel é um cliente da Autoridade Certificadora 3. A empresa de automóveis precisa, agora, conseguir o certificado da AC 3. Esse certificado pode ser conseguido através de um e-mail, ou no site da AC 3, ou de outra forma que a AC 3 o disponibilizar. Pronto, a empresa de Automóveis pode conferir agora a veracidade do certificado do cliente.

Primeiro, a empresa de Automóveis utiliza o certificado digital da AC 3 para conferir se o certificado do cliente foi assinado por aquela AC. Confirmando isso, a empresa agora irá utilizar o certificado da AC Raiz para verificar se o certificado da AC 3 foi assinado pela própria Raiz. Foi feito assim, toda a validação da cadeia hierárquica até a AC Raiz. Por isso a importância do certificado da AC Raiz ser extremamente confiável. Porque a partir desse certificado, é possível verificar se algum certificado abaixo dela foi alterado ou se foi gerado por outra AC, não confiável. Então, passando por estas etapas, o cliente será autorizado a efetivar a compra.

Todo esse procedimento pode ser reduzido. Uma vez que o certificado da AC 3 for validado e garantido pelo certificado da AC Raiz, não será necessário refazer essa validação toda vez que um cliente tentar efetuar uma compra. Basta guardar o certificado da AC 3 de forma segura e então a empresa irá validar o certificado dos cliente apenas com a AC 3, não precisando mais passar pela AC Raiz. É importante destacar que todo esse procedimento descrito é feito de forma transparente e automática para o cliente.[6]

3.1.2 - Arquitetura Mista

Esta arquitetura, também conhecida como certificação cruzada, múltiplas AC's se validam em um ambiente. Cada usuário confia em uma única AC, mas as AC's não se reportam unicamente a uma AC superior a ela, como acontece na arquitetura hierárquica. Na visão do usuário isso é transparente. Afinal, ele só confia na AC que assinou o seu certificado. Mas para as AC's isso muda.

O diagrama abaixo representa um exemplo desta arquitetura.

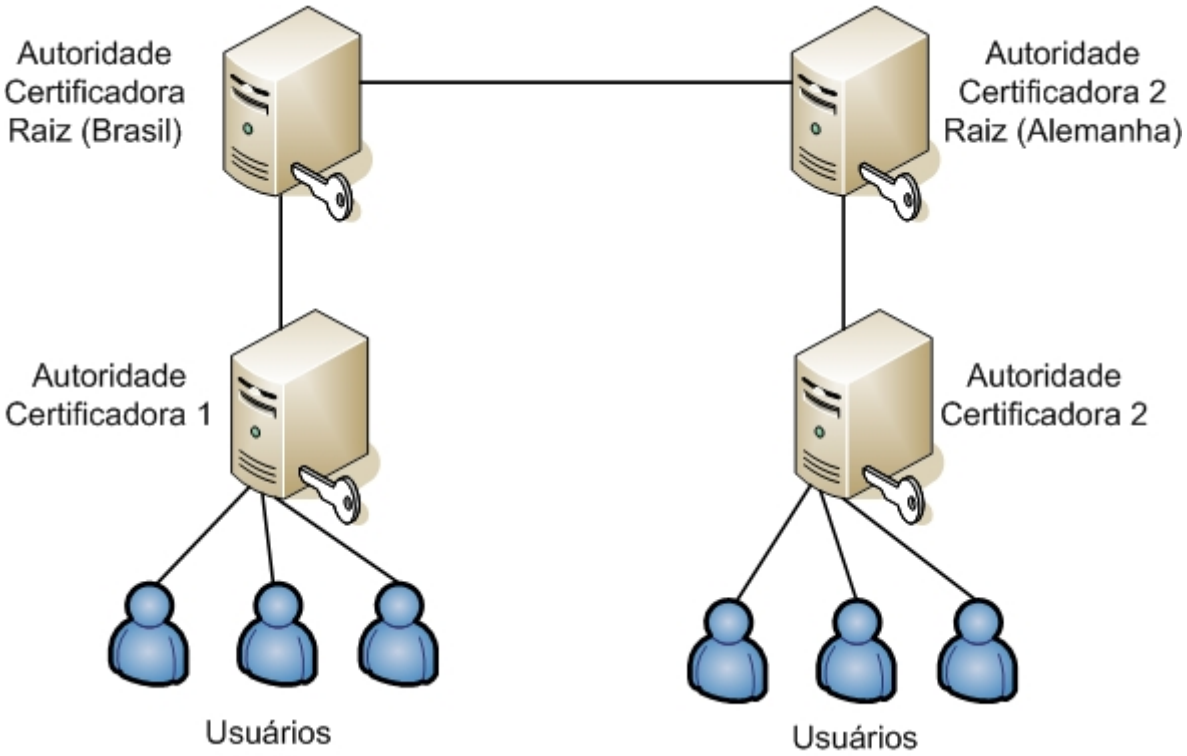

Figura 8 - Arquitetura Mista

No exemplo acima, pode ser observado que ainda existe a estrutura hierárquica citada anteriormente, mas desta vez, a AC Raiz não é o ultimo ponto desta rede. As AC's Raízes podem confiar em outras AC's Raízes, criando um ciclo de confiança e não limitando a sua abrangência apenas a sua hierarquia. Contextualizando, digamos que um indivíduo brasileiro esteja de férias na Alemanha. Essa pessoa aluga um carro alemão, gosta do modelo e do estilo do carro e decide trazer um para o Brasil. Ele se dirige para uma revendedora deste modelo de

carro e solicita um exemplar. E que este seja enviado ao Brasil. A revendedora exige do cliente para isso, o seu certificado digital e o número de sua conta corrente. Com isso, ela poderá efetivar a compra.

O cliente, então, entrega um certificado digital assinado por uma AC brasileira, situada abaixo da AC Raiz do Brasil. A revendedora irá validar a cadeia de assinaturas. Quando isso ocorrer, irá perceber que aquele certificado não está abaixo na hierarquia da AC Raiz da Alemanha. Mas a AC Raiz da Alemanha possui uma confiança na AC Raiz do Brasil. Com isso, informa a revendedora que aquele certificado em questão é válido e confiável, já que foi assinada por uma AC Raiz conhecida. O cliente, então, poderá efetivar a sua compra.

É necessário lembrar sempre que esse procedimento é transparente para o cliente e que essa arquitetura pode ser montada com uma quantidade variável de AC's Raízes.[6]

3.2 - Certificados digitais para usuários (Modelo X.509)

Os certificados digitais são um meio seguro de distribuir as chaves públicas dentro de uma rede. São análogos a um CPF ou uma carteira de identidade, por exemplo. Em qualquer caso, existe um terceiro confiável que confirma a identidade e os privilégios e direitos de acesso do usuário. Em seu padrão mais básico, um certificado digital contém uma chave pública, a identidade da pessoa a qual ele pertence e o nome da parte que estará atestando a validade deste.

Diversos certificados estão em utilização. Alguns dele, como o Pretty Good Privacy (PGP), são patenteados. Outros certificados são específicos de um aplicativo como certificados SET e o Internet Protocol Security (IPSec). O padrão de certificado digital mais aceito atualmente no mercado é o X.509 Versão 3 da ITU (International Telecommunications Union). Esse padrão de certificados, apesar de ser amplamente usado para a Internet, é também muito utilizado em ambientes empresariais.

Figura 9 - Certificado Digital padrão X.509 (Certificado raiz da ICP-Brasil)

A tabela abaixo representa a disposição dos campos de um certificado padrão X.509 v3.

Tabela 1 – Estrutura do X.509 v3

Version **(Versão)**
Certificate Serial Number **(Número Serial do Certificado)**
Signature Algorithm Identifier **(Identificador do algoritmo de assinatura)**
Issuer Name **(Nome do emissor)**
Validity (Not Before/Not After) **Validade(Não antes/Não depois)**
Subject Name **(Nome do sujeito)**
Subject Public Key Information **(Informações da chave pública do sujeito)**
Issuer Unique Identifier **(Identificador único de emissor)**
Subject Unique Identifier **(Identificador único do sujeito)**
Extensions **(Extensões)**
Signature **(Assinatura)**

Version (Versão): Esse campo define a versão do certificado, como Versão 1, Versão 2, Versão 3. Esse campo permite possíveis versões futuras.

Certificate Serial Number (Número serial do certificado): Esse campo contém um valor único em cada certificado gerado pela AC.

Signature Algorithm Identifier (Identificador do algoritmo de assinatura): Indica o identificador do algoritmo utilizado para assinar o certificado.

Issuer Name (Nome do emissor): Esse campo identifica o nome distinto (Distinguished Name-DN) com o qual a AC cria e assina esse certificado.

Validity (Validade(Não antes/Não depois)): Contém dois valores de data/hora que definem o período que esse certificado pode ser considerado válido a menos que seja revogado.

Subject Name (Nome do sujeito): Esse campo identifica o DN da entidade final a que o certificado se refere, ou seja, o possuidor da chave privada correspondente.

Subject Public Key Information (Informações da chave pública do sujeito): Esse campo possui o valor da chave pública do sujeito, bem como o identificador de algoritmo e qualquer outro parâmetro associado ao algoritmo pelos quais a chave deve ser utilizada.

Issuer Unique Identifier (Identificador único de emissor): Campo opcional que contém um identificador único que é utilizado para exibir de maneira não-ambígua o nome da AC, em casos onde um mesmo nome foi reutilizado por diferentes entidades ao longo do tempo.

Subject Unique Identifier (Identificador único do sujeito): Campo opcional que contém um identificador único que é utilizado para exibir de maneira não-ambígua o nome do proprietário do certificado, em casos onde um mesmo nome foi reutilizado por diferentes entidades ao longo do tempo.

Extensões de Certificado X.509 Versão 3

As versões 1 e 2 de certificados X.509 ainda possuíam algumas deficiências. Muitos dados importantes não eram inseridos no certificado por causa do padrão adotado. Por essa razão, a versão 3 veio com uma alteração importante que era o conjunto de extensões. Essas extensões abrangem as informações sobre a política, a chave, os atributos do sujeito e do emissor e as restrições do caminho de certificação. As informações contidas nas extensões podem ser marcadas como críticas ou não-críticas. Os seguintes campos definem as extensões dos certificados digitais X.509 v3.

Authority Key Identifier (Identificador de chave de autoridade): Este campo é utilizado para diferenciar chaves de assinaturas com múltiplos certificados de uma mesma AC. A AC fornece um identificador único de chave ou um ponteiro para um outro certificado.

Subject Key Identifier (Identificador de chave do sujeito): Essa extensão é utilizada para diferenciar chaves de assinaturas com múltiplos certificados de um mesmo proprietário de certificado. O proprietário fornece um identificador único de chave ou um ponteiro para um outro certificado.

Key Usage (Utilização da chave): Essa extensão é utilizada para definir as restrições quanto às operações que podem ser realizadas pela chave pública do certificado, como assinatura digital, assinatura de certificado, cifragem de dados, etc.

Extended Key Usage (Utilização de chave estendida): Pode ser utilizado além do campo Key Usage para definir uma ou mais utilizações da chave pública do certificado, como utilização do protocolo TLS.

CRL Distribution Point (Ponto de Distribuição de CRL): Esse campo apresenta um identificador de recurso uniforme (URI) para localizar a estrutura de CRL definida.

Private Key Usage Period (Período de uso de chave privada): Semelhante ao campo Validity, essa extensão indica o período de tempo para utilização da chave privada associada à chave pública nesse certificado.

Certificate Policies (Políticas de Certificado): Essa extensão identifica as informações sobre as políticas e qualificadores opcionais que a AC associa ao certificado.

Policy Mappings (Mapeamento de políticas): Utilizada apenas quando o sujeito do certificado também for uma AC.

Subject Alternative Name (Nome alternativo do sujeito): Indica uma ou mais formas alternativas de nomes associados ao proprietário desse certificado. Podem ser incluídos nomes diversos, como o próprio e-mail.

Issuer Alternative Name (Nome alternativo do emissor): Indica uma ou mais formas alternativas de nomes associados ao emissor do certificado.

Subject Directory Attributes (Atributos do diretório do sujeito): Fornece as informações adicionais sobre a identidade do sujeito que não são transportadas para os campos de nome, como telefone, posição dentro da empresa, etc.

Basic Constraints (Restrições Básicas): Indica se o sujeito pode agir como uma AC, fornecendo uma maneira para restringir que usuários finais atuem como AC's.

Name Constraints (Restrições de nomes): Deve ser utilizada apenas dentro de certificados de AC. Ela especifica o espaço de nome dentro do qual todos os nomes de sujeito devem estar localizados para qualquer certificado subseqüente que seja parte desse caminho de certificado.[7]

A figura 10 representa um certificado X.509 e alguns de seus campos apresentados.

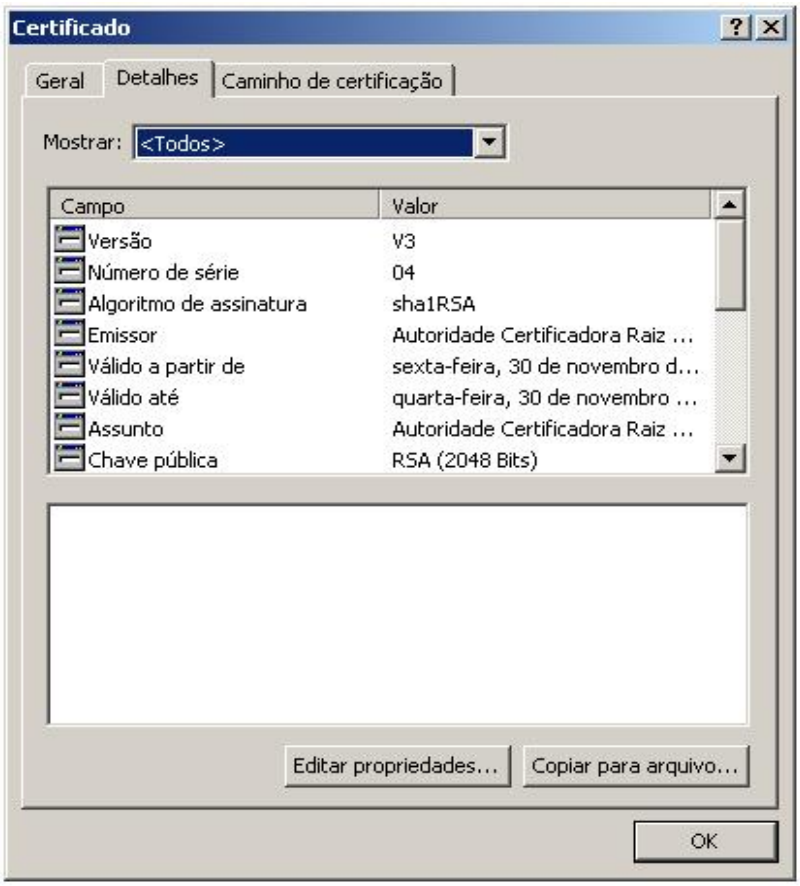

Figura 10 - Certificado Digital X.509

3.3 – Protocolos PKCS

Alguns protocolos foram criados para se utilizar e implementar uma transação de ICP. Definidos como PKCS (Public Key Cryptography Standards), esses padrões foram criados pela RSA Security em conjunto com outras grandes empresas para auxiliar a utilização de uma infra-estrutura de chaves públicas e padronizar essa utilização no mercado.

A tabela 2 lista os padrões PKCS ainda ativos.

Tabela 2 – Definição dos protocolos PKCS

Padrão	Descrição
PKCS #1	Esse padrão define mecanismos para cifragem e assinatura de dados usando o sistema RSA.
PKCS #3	Define o padrão de chaves Diffie-Hellman.
PKCS #5	Descreve um método para gerar uma chave secreta baseado em uma senha.
PKCS #6	Padrão de sintaxe de certificado.
PKCS #7	Define uma sintaxe genérica para mensagens que devem ser criptografadas. Esse protocolo será melhor definido em seguida.
PKCS #8	Define um método para a guarda de informações de chave privada.
PKCS #9	Define tipos de atributos para uso nos protocolos PKCS.
PKCS #10	Padrão de Requisição de Certificados.
PKCS #11	Padrão de interface para criptografia em Tokens.
PKCS #12	Descreve um formato portátil para guarda e transporte de chaves privadas, certificados, etc.
PKCS #13	Esse padrão define mecanismos para cifragem e assinatura de dados usando o algoritmo de curvas elípticas.
PKCS #14	Padrão para geração de números pseudo-randômicos.
PKCS #15	Descreve um padrão para informações credenciais criptografadas armazenadas em tokens.

Os protocolos PKCS #7 e #10 são os mais utilizados atualmente. Os padrões PKCS #2 e #4 não mais existem, pois foram incorporados no PKCS #1. [8]

3.3.1 - PKCS #10

O protocolo PKCS #10 é muito utilizado atualmente. Ele representa o padrão de Requisição de Certificados. Esse padrão possui um formato que agrupa um DN (Distinguished Name), uma chave pública, um campo opcional de atributos, um identificador de algoritmo e uma assinatura digital. O campo opcional foi designado para complementar atributos para inclusão no certificado digital, como por exemplo, e-mail.

Essa requisição deve ser assinada utilizando a chave privada do solicitante. Dessa forma, ao enviar o PKCS #10 para a AC, esta terá a certeza de que o solicitante é possuidor da chave privada relativa àquela chave pública enviada no PKCS #10. Utilizando a chave pública contida no PKCS #10, a AC irá aplicar essa chave na requisição recebida e verificará se a chave pública que recebeu é o complemento da chave particular que gerou o CSR (Requisição de assinatura de certificado).[9]

A tabela abaixo representa os campos do padrão PKCS # 10.

Tabela 3 - Protocolo PKCS #10

Versão
DN - Distinguished Name
Chave Pública
Atributos (Opcional)
Algoritmo de Assinatura
Assinatura Digital

3.3.2 - PKCS #7

O padrão PKCS #7 descreve uma sintaxe genérica para dados que podem ser submetidos a funções criptográficas, tais como assinatura e envelopagem digital. Permite recursividade, com aninhamento de envelopes. Permite também a associação de atributos arbitrários, como, por exemplo, contra-assinatura. Esse padrão pode dar suporte a uma variedade de arquiteturas de gerenciamento baseadas em infra-estrutura de chaves públicas.

Esse protocolo provavelmente é o mais utilizado de todos devido a sua característica genérica para utilização de chaves públicas. Foi criado especificamente para proteger

mensagens com criptografia mas pode ser utilizado de outras formas. Esse padrão define basicamente dois campos: o **tipo de conteúdo** e o **conteúdo**. O campo do **tipo de conteúdo** pode ser classificado em seis tipos diferentes: dados (**data**), dado assinado (**signed data**), dado envelopado (**enveloped data**), dado assinado e envelopado (**Signed and Enveloped Data**), HASH do dado assinado (**Digested data**) e dado encriptado (**Encrypted Data**). Para melhor compreensão do PKCS #7, seus tipos primitivos serão detalhados.

Dados (Data): Essa definição indica que o protocolo irá guardar apenas o dado em questão, podendo, por exemplo, ser um texto em ASCII ou mesmo uma informação binária.

Dado Assinado (Signed Data): Essa definição caracteriza o protocolo como um tipo assinado. Ou seja, o padrão armazena uma assinatura de um dado qualquer. Esse tipo pode conter também a própria mensagem que foi assinada.

Dado Envelopado (Enveloped Data): Esse padrão consiste em uma estrutura de dados contendo informações encriptadas de qualquer tipo e a chave simétrica (utilizada para a cifragem da mensagem) encriptada pela chave pública do destinatário da mensagem O processo pelo qual um envelope digital é formado envolve os seguintes passos:

1 – Uma chave de encriptação simétrica é gerada aleatoriamente.

2 – Utiliza-se essa chave para encriptar a mensagem.

3 – Em seguida, a chave pública do destinatário é utilizada para cifrar a chave simétrica.

4 – Ao final, o padrão possui uma mensagem cifrada simetricamente e uma chave simétrica cifrada com uma chave pública.

Dado Assinado e Envelopado (Signed and Enveloped Data): Esse tipo consiste em uma estrutura contendo dados encriptados, a chave simétrica utilizada para cifrar os dados em questão, a chave pública de destino e um HASH assinado. O processo pelo qual um tipo **Assinado e Envelopado** é formado envolve os seguintes passos:

1 – Uma função HASH é aplicada sobre os dados a serem enviados.

2 – A chave privada do remetente é aplicada sobre esse HASH gerado, formando assim, a assinatura dos dados.

3 – Utiliza-se uma chave simétrica para cifrar os dados em questão.

4 – A chave pública do destinatário é usada para cifrar essa chave simétrica gerada anteriormente.

5 – Ao final, o PKCS #7 possui uma mensagem cifrada simetricamente, um HASH criptografado por uma chave privada e uma chave simétrica criptografada com a chave pública do destinatário.

HASH do Dado Assinado (Digested Data): Esse tipo padroniza o PKCS #7 com um HASH da mensagem e a assinatura desse HASH, podendo ou não conter a própria mensagem assinada. Ou seja, um HASH é aplicado sobre a mensagem, gerando uma cadeia de bytes. Em seguida, a chave privada é aplicada sobre o resultado desse HASH. O protocolo irá conter exatamente esse HASH cifrado pela chave privada.

Dado Encriptado (Encrypted Data): Nesse padrão, o PKCS #7 transporta apenas a mensagem cifrada simetricamente. Pressupoe-se que a chave simétrica já tenha sido trocada de forma segura anteriormente. Por questões de segurança, não se envia a chave simétrica em aberto para o destinatário.[10]

3.4 - ICP Brasil e políticas de segurança de uma AC

Este tópico tem como objetivo apresentar de forma clara uma breve definição do que vem a ser o órgão ICP-Brasil, que rege todas as leis referentes a infra-estruturas de chaves públicas no Brasil. Esse instituto deve ser apresentado, pois todo o projeto teve como base essas normas. O tópico relativo à segurança de uma AC foi enfatizado devido a importância dessa segurança. Qualquer falha em uma Autoridade Certificadora pode comprometer toda a infra-estrutura.

O **Instituto Nacional de Tecnologia da Informação – ITI**, autarquia federal vinculada à Casa Civil da Presidência da República, é a **Autoridade Certificadora Raiz – AC Raiz** da Infra-estrutura de Chaves Públicas Brasileira – ICP-Brasil.

Como tal é a primeira autoridade da cadeia de certificação, executora das Políticas de Certificados e normas técnicas e operacionais aprovadas pelo Comitê Gestor da ICP-Brasil, e tem por competências emitir, expedir, distribuir, revogar e gerenciar os certificados das Autoridades Certificadoras - AC de nível imediatamente subseqüente ao seu; gerenciar a lista de certificados emitidos, revogados e vencidos; executar atividades de fiscalização e auditoria

das AC, das Autoridades de Registro - AR e dos prestadores de serviço habilitados na ICP-Brasil. Seu certificado constitui um Certificado Auto-Assinado, já que é o primeiro de toda a hierarquia.

É de responsabilidade ainda do ITI estimular e articular projetos de pesquisa científica e de desenvolvimento tecnológico voltados à ampliação da **cidadania digital**. Neste setor, o ITI tem como sua principal linha de ação a popularização da certificação digital e a inclusão digital, atuando sobre questões como sistemas criptográficos, software livre, hardware compatíveis com padrões abertos e universais, convergência digital de mídias, entre outras.

As políticas de certificados descritas na tabela abaixo são definidas pela ICP-Brasil. É uma forma de se qualificar o certificado de acordo com a forma de geração ou armazenamento do mesmo. Além disso, a nomenclatura sugere o uso do certificado, Assinatura (A) ou Sigilo (S). Certificados de tipos A1, A2, A3 e A4 são utilizados em aplicações como confirmação de identidade na *Web*, correio eletrônico, transações *on-line*, redes privadas virtuais, cifragem de chaves de sessão e assinatura de documentos eletrônicos com verificação da integridade de suas informações. Certificados de tipos S1, S2, S3 e S4 serão utilizados em aplicações como cifragem de documentos, bases de dados, mensagens e outras informações eletrônicas, com a finalidade de garantir o seu sigilo.

A tabela abaixo descreve os tipos de certificados definidos pela ICP-Brasil.

Tabela 4 - Tipos de certificados

Tipo de Certificado	Chave	Geração do Par de Chaves	Criptografia	Armazenamento do Certificado e da Chave Privada	Validade
A1 e S1	1024	Software	Software	Software	1 ano
A2 e S2	1024	Software	Software	Hardware	2 anos
A3 e S3	1024	Hardware	Hardware	Hardware	3 anos
A4 e S4	2048	Hardware	Hardware	Hardware	3 anos

O projeto permite a utilização do tipo A1, podendo ser implementado futuramente a utilização dos outros tipos.

Atualmente existem diversas AC's situadas abaixo na cadeia hierárquica da ICP Brasil. Essas AC's são responsáveis por gerar certificados digitais, revogar, divulgar listas de certificados revogados, entre outras funções. A estrutura abaixo representa a hierarquia atual da ICP-Brasil.

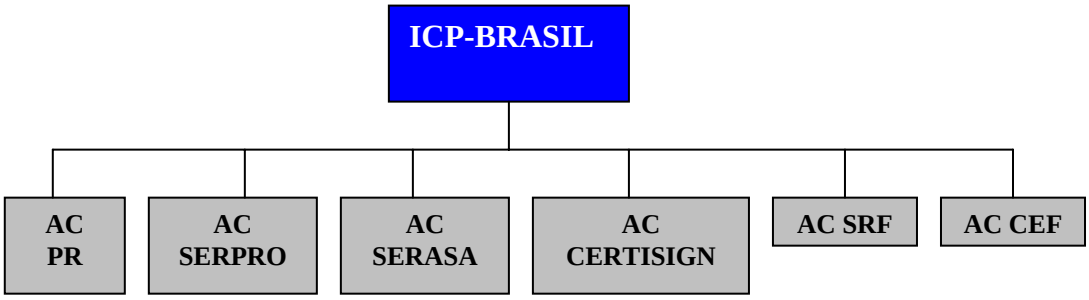

Figura 11 - Hierarquia da ICP-Brasil

Cada AC possui sua aceitação dentro de uma política de certificados definida. O quadro abaixo apresenta cada AC na ICP-Brasil e suas definições de certificados com que trabalha.[11]

Tabela 5 - Políticas de certificados das AC's da ICP-Brasil

AUTORIDADE CERTIFICADORA	POLÍTICAS DE CERTIFICADO
AC Presidência da República	A1, A3
AC SERPRO	A1, A3, SPB
AC SERASA	AC
AC SERASA	SPB
AC SERASA CD	A1, A2, A3, A4 S1, S2, S3, S4
AC CERTISIGN	AC
AC CERTISIGN SPB	SPB
AC CERTISIGN MULTIPLA	A1, A2, A3, A4 S1, S2, S3, S4
AC SERTISIGN SRF	A1, A3, A4
AC SRF	AC
AC SERPRO SRF	A1, A3
AC SERASA SRF	A1, A3, A4
AC CAIXA	AC
AC CAIXA PF	A1, A3
AC CAIXA PJ	A1, A3
AC CAIXA IN	A1, A3

3.4.1 - Segurança Física de uma AC

A segurança necessária a uma AC é de extrema importância. Uma única falha nesse processo pode comprometer toda a estrutura de chaves públicas em que esta AC esteja envolvida. Por exemplo, se um indivíduo de alguma forma conseguir invadir a base de dados de uma AC e, dessa forma, roubar a chave privada da mesma, essa pessoa poderá gerar certificados digitais e assiná-los com a chave privada que roubou. Dessa forma, esse indivíduo poderá se passar por qualquer pessoa ou até mesmo pela AC de quem roubou a chave privada. Mas uma infra-estrutura de chaves públicas tem sua proteção em caso de exposição da chave privada. Caso isso venha a acontecer, a AC avisa a todos os participantes da ICP-Brasil o ocorrido, envia um CSR a AC acima dela na hierarquia e recebe um novo certificado. Em seguida, distribui novamente o seu novo certificado pela infra-estrutura de chaves públicas.

Essa segurança exigida pela ICP é muito ampla e cara, mas necessária. Abrange tópicos que poucas empresas sequer imaginaram abordar em sua segurança física. Segue abaixo uma relação de algumas preocupações que uma AC deve ter com seu ambiente físico.

- Sistemas de segurança deverão ser instalados para controlar e auditar o acesso aos sistemas de certificação.
- Os sistemas de AC deverão estar localizados em área protegida ou afastados de fontes potentes de magnetismo ou interferência de rádio freqüência.
- Recursos e instalações críticas ou sensíveis devem ser mantidos em áreas seguras, protegidas por um perímetro de segurança definido, com barreiras de segurança e controle de acesso. Elas devem ser fisicamente protegidas de acesso não autorizado, dano, ou interferência. A proteção fornecida deve ser proporcional aos riscos identificados.
- Quaisquer equipamentos de gravação, fotografia, vídeo, som ou outro tipo de equipamento similar, só devem ser utilizados a partir de autorização formal e mediante supervisão.
- Sistemas de detecção de intrusos devem ser instalados e testados regularmente de forma a cobrir os ambientes, as portas e janelas acessíveis, nos ambientes onde ocorrem processos críticos. As áreas não ocupadas devem possuir um sistema de alarme que permaneça sempre ativado.

Existem AC's que possuem salas cofres com vários níveis de acesso, sendo que uma única pessoa não possui acesso em todas as portas dos níveis. Ou seja, dessa forma, sempre é preciso de mais de um indivíduo para se atingir o último nível, dificultando a fraude em uma

AC. Até mesmo os desastres causados por fogo ou água (como um sistema anti-fogo) devem ser reestruturado.

As auditorias da ICP para validar uma instituição como uma AC são sempre muito rigorosos, sendo que diversas vezes, são necessárias muitas auditorias até que se consiga um título de AC. [12]

4 – Uma abordagem de ICP para ambientes corporativos

Este capítulo do trabalho refere-se à implementação do aplicativo, sua instalação e utilização e algumas políticas de segurança, visando o desenvolvimento de uma infra-estrutura de chaves públicas em um ambiente coorporativo, permitindo a troca segura e confiável de informações dentro de uma empresa, garantindo confidencialidade, autenticidade e não-repúdio das informações enviadas. A própria empresa irá possuir uma Autoridade Certificadora, gerando certificados para seus funcionários e, se desejar, para seus clientes e parceiros comerciais, reduzindo o custo de se implementar uma estrutura de chaves públicas já preparada por uma AC comercial.

Este trabalho teve uma influência mercadológica muito grande. Atualmente as corporações passam por muitos problemas relacionados a fraudes eletrônicas.
Algumas delas são:

- Dados enviados em claro, trafegando pela rede sem nenhuma proteção. Esses dados podem ser capturados e lidos enquanto trafegam pela empresa. Podem se tratar de projetos ou informações cruciais que não deveriam ser lidos por qualquer funcionário, podendo comprometer a empresa.
- Informações que são enviadas a um destinatário e são alteradas antes de chegar no mesmo. Por exemplo, um documento enviado pelo diretor financeiro da empresa para a área de compra com um pedido de 100 unidades de determinada impressora pode ser facilmente alterado para 1000 unidades, provocando um prejuízo na empresa.
- Documentos que são enviados em nome de uma pessoa sendo que ela nem mesmo tem o conhecimento da existência destes. Uma solicitação com o e-mail e o nome do presidente pode ser enviada a área de informática da empresa para se alterar alguma configuração de rede.

Estas fraudes causam grande prejuízo a algumas empresas. Este projeto permite que esse tipo de fraude seja evitado, minimizando os prejuízos de uma empresa e garantindo segurança nas informações que precisam ser trafegadas pela rede. O aplicativo pode também garantir a guarda de dados. Apesar de ser um objetivo secundário, esse projeto pode também fazer essa função. Um bom meio de se fazer isto é criptografando o dado com a chave pública de um funcionário e guardar a informação. Caso essa informação precise ser resgatada, utiliza-se a chave privada do mesmo funcionário. Então, cada indivíduo dentro da empresa

pode guardar seus dados em seu computador de forma segura, garantindo a confidencialidade do documento.

Os benefícios que o projeto pretende prover podem ser definidos de forma simples. Confidencialidade, autenticidade e não-repúdio dos dados.

Através da criptografia assimétrica, essa infra-estrutura irá garantir a segurança e a autenticidade dos dados de uma empresa. As informações não mais irão trafegar em claro, impedindo que terceiros leiam dados que não deveriam ser lidos. Além do que pode-se assinar digitalmente um documento, garantindo que o remetente é realmente a pessoa e não um falsário se passando por ela.

Além disso, este trabalho irá trazer para a corporação uma economia em gastos que seriam feitos com Autoridades Certificadoras externas ou firewalls e outros mecanismos que não ajudariam muito neste problema, já que a visão do projeto é impedir alguns tipos de ataques internos. E também irá minimizar o prejuízo da empresa com fraudes eletrônicas.

Uma boa política de segurança e um aplicativo que gerencie e manipule todo o trabalho com criptografia assimétrica pode ter resultados de forma rápida e confiável, refletindo essa segurança nos orçamentos da empresa.

4.1 Fluxograma

O projeto desenvolvido pode ser muito bem representado por fluxogramas que demonstrarão cada processo executado em toda a infra-estrutura de chaves públicas. Todos os passos serão muito detalhados para que o entendimento seja o mais claro possível.

O fluxograma do projeto foi dividido em quatro etapas que são os principais processos que ocorrerão nessa estrutura de segurança desenvolvida. Cada etapa será explicada e ilustrada de forma a facilitar o entendimento de toda a infra-estrutura e auxiliando também no entendimento do aplicativo.

4.1.1 – Geração do Certificado Digital

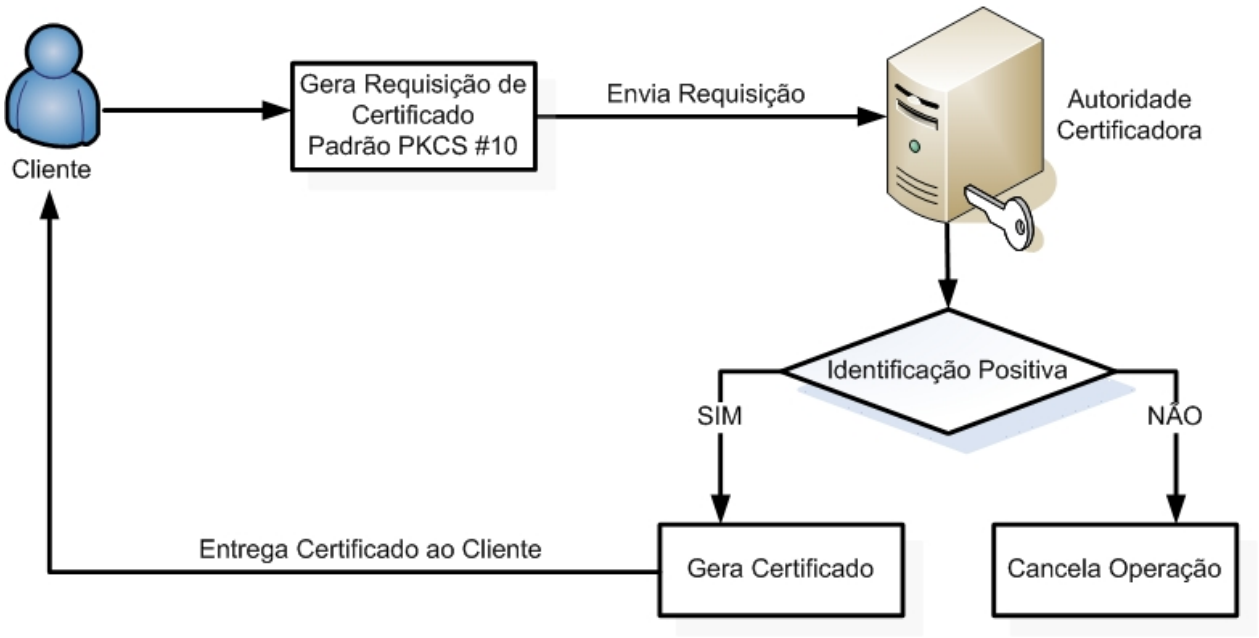

Figura 12 - Fluxograma de geração de certificado

Esse fluxograma representa a geração de um certificado digital. O cliente inicia todo o procedimento gerando uma CSR (Requisição de Certificado). Nessa geração de CSR, o cliente irá gerar o par de chaves criptográficas e um arquivo CSR no padrão PKCS #10 explicado anteriormente. Em seguida, o cliente deve se dirigir a Autoridade Certificadora para se identificar. A AC, após verificar a identidade do cliente, irá gerar e assinar um certificado tornando-o válido perante a infra-estrutura de chaves públicas.

4.1.2 – Troca de chaves criptográficas

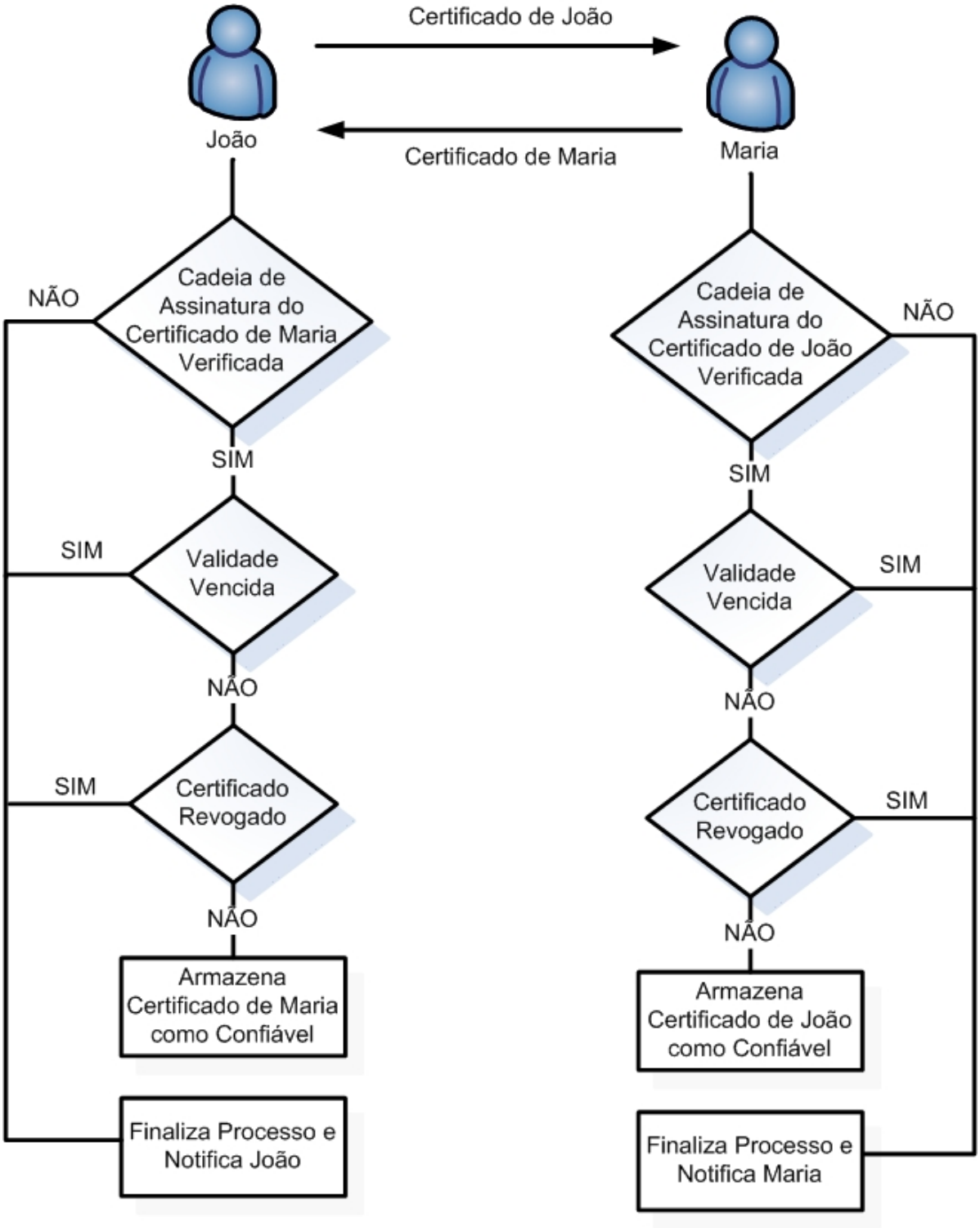

Figura 13 - Fluxograma de troca de chaves criptográficas

Este fluxograma representa a troca de chaves criptográficas do projeto. Para que haja uma interação dos usuários dentro de uma infra-estrutura de segurança, estes usuários

precisam trocar suas chaves públicas de criptografia. A figura acima será detalhada para um melhor entendimento deste processo.

Exemplificando, o funcionário João envia o seu certificado para Maria. Maria, ao receber o certificado de João, precisa ter a certeza de que esse certificado é válido e conferido pela Autoridade Certificadora em questão.

Então, a cadeia de assinatura do certificado de João é verificada. Nesse procedimento, será possível determinar se a AC assinou esse certificado digital ou se este certificado foi gerado por uma outra AC não confiável, tornando o certificado inválido. A validade do certificado é conferida em seguida, pois um certificado vencido não pode ser utilizado para não comprometer a segurança da estrutura projetada.

No procedimento de verificação de certificados revogados, Maria consegue a Lista de Certificados Revogados com a AC para verificar se o certificado que ela recebeu de João não foi cancelado por algum motivo.

Ao final de todas essas verificações, o certificado pode ser considerado confiável, pois não está revogado, nem vencido e Maria tem a certeza de que foi assinado pela AC em que ela confia. Este certificado recebe o status de confiável. Todo esse procedimento é transparente para o usuário.

4.1.3 – Envio de mensagens

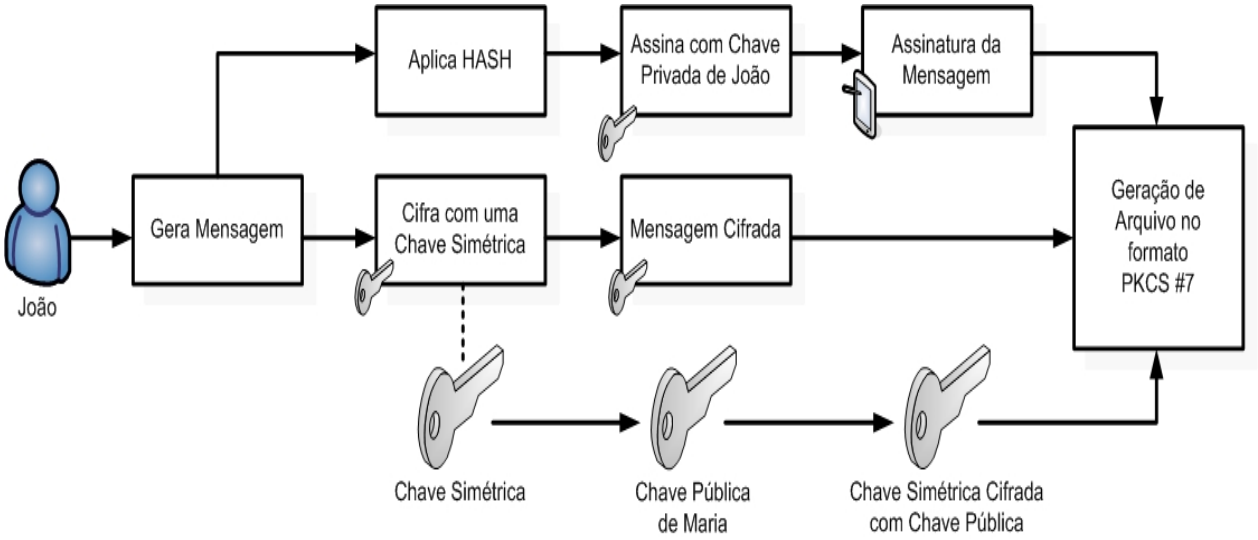

Figura 14 - Fluxograma de geração de mensagens

A troca de mensagens entre usuários da infra-estrutura é o procedimento mais complexo envolvido. Uma explicação mais detalhada é necessária para melhor se compreender todo esse processo.

João irá gerar uma mensagem para Maria que contém as características de uma mensagem assinada e envelopada.

Com a mensagem já definida, João irá aplicar um algoritmo de HASH gerando um resumo da mensagem que ele quer enviar. Em seguida, utiliza sua **chave privada** para assinar esse resumo que foi gerado. Então, ele tem agora um resumo assinado da mensagem.

Em paralelo, ele utiliza uma **chave simétrica** para poder criptografar a mensagem que ele quer enviar. Ao final deste processo, ele possui agora uma mensagem criptografada por uma chave simétrica. Mas para Maria conseguir abrir essa mensagem, ela também precisa da mesma chave simétrica que cifrou a mensagem. Para enviar essa chave em segurança, João aplica a **chave pública de Maria** para cifrar essa chave simétrica.

Em posse de um HASH assinado, uma mensagem criptografada e uma chave simétrica cifrada com a chave pública do destinatário, João irá gerar um arquivo no padrão PKCS #7 contendo estes dados que serão enviados para Maria.

4.1.4 – Recebimento de mensagens

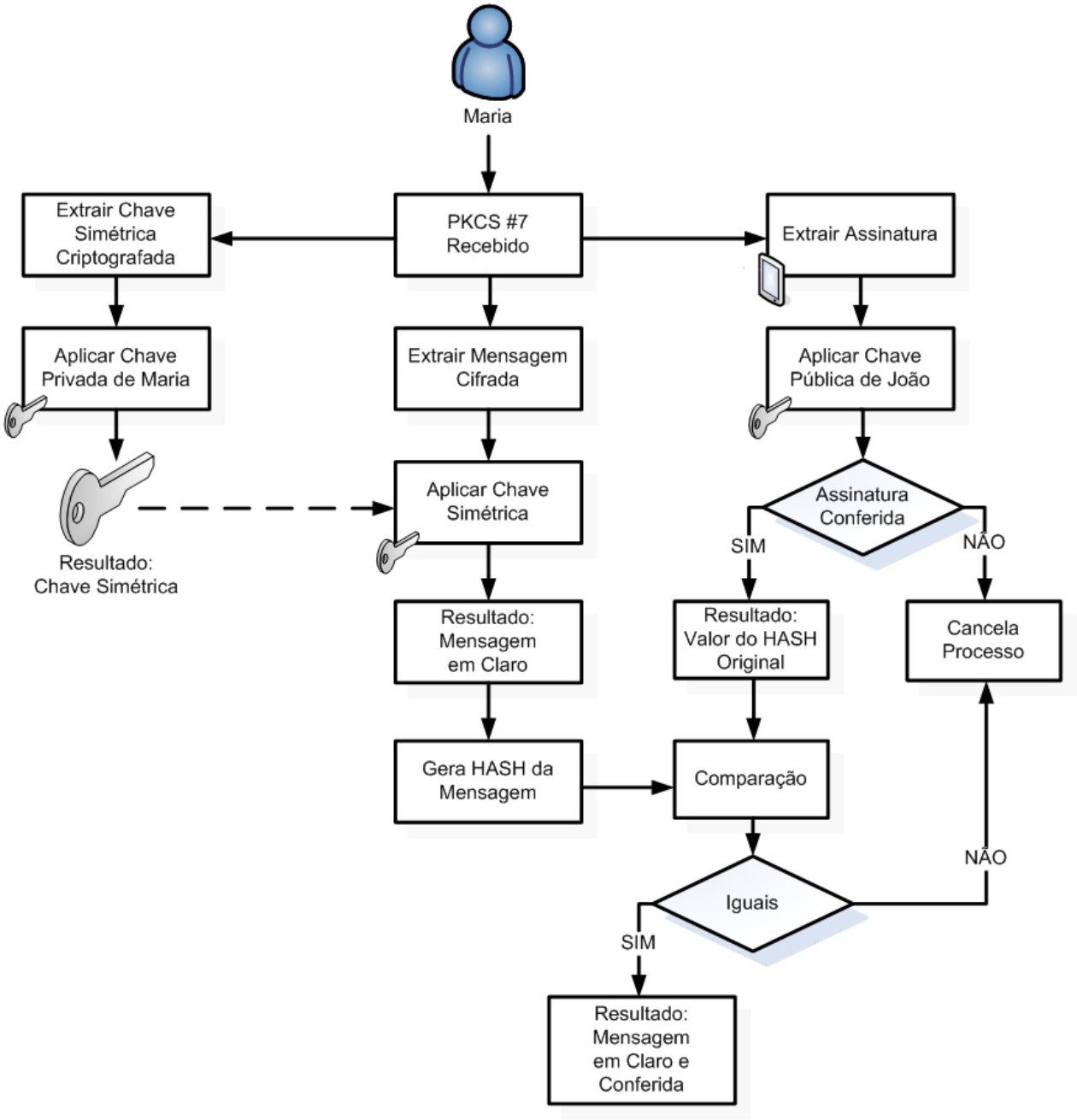

Figura 15 - Fluxograma de recebimento de mensagens

A etapa de recebimento de mensagens completa o circulo de procedimentos de utilização da infra-estrutura projetada. Esse procedimento se inicia com Maria recebendo o arquivo no padrão PKCS #7.

Ao receber esse arquivo, Maria precisa desfazer todo o processo feito anteriormente. Se nenhum problema for apresentado na inversão do processo, Maria poderá ler a mensagem

que lhe foi enviada, tendo a certeza de que não foi alterada e que ninguém mais poderia ter lido a informação contida no arquivo.

O primeiro procedimento a seguir é o de se extrair a assinatura que está inserida no arquivo que João enviou. Como a assinatura consiste em um HASH que se aplicou a chave privada de João, Maria irá aplicar a chave pública de João que pode ser conseguida no próprio certificado digital de João que ela recebeu anteriormente. Maria agora possui o valor do HASH original que foi gerado da mensagem por João. Ela irá precisar deste dado futuramente.

Em seguida, Maria extrai a chave simétrica que foi criptografada pela sua chave pública. Ela utiliza a sua chave privada e consegue desfazer o que sua chave pública cifrou, conseguindo a chave simétrica que será utilizada para decifrar a mensagem que está cifrada.

Agora Maria irá extrair a mensagem cifrada. Para conseguir ler a mensagem cifrada, Maria irá utilizar a chave simétrica que acabou de decifrar para desfazer a criptografia aplicada à mensagem. Nesse momento, Maria já conseguiu algumas garantias a respeito de se utilizar a infra-estrutura de chaves públicas. Como ela, utilizando a chave pública de João, conseguiu desfazer a assinatura e conseguir o HASH (que foi assinado pela chave privada de João), já tem a certeza de que foi João quem assinou essa mensagem. Também, como sabe que a chave simétrica que cifrou a mensagem esta protegida pela sua chave pública e que somente ela pode desfazer esta criptografia, ela tem a certeza de que ninguém mais leu esta mensagem.

Só resta saber se após João assinar a mensagem, esta não foi alterada. Para ter essa certeza, Maria aplica o mesmo algoritmo de HASH que João utilizou para se conseguir o valor do resumo. Após conseguir esse valor, ela compara com o valor original que veio com o arquivo PKCS #7 e verifica se eles são iguais. Caso os valores sejam idênticos, significa que essa mensagem não foi alterada em nenhum momento. Isso acaba de garantir para Maria toda confidencialidade, autenticidade e não-repúdio que uma infra-estrutura de segurança poderia oferecer.

4.2 – Políticas de Segurança

Essa política tem como objetivo orientar todas as ações de segurança, para reduzir riscos e garantir a integridade, sigilo e disponibilidade das informações dos sistemas de informação e recursos.

A política de segurança necessária para a implementação e utilização do projeto deve ser rigorosamente seguida. Uma ocorrência de fraude pode comprometer um usuário ou, até mesmo, toda a infra-estrutura de segurança de uma empresa. Essa segurança se estende por todos os usuários desse serviço e, principalmente, a Autoridade Certificadora, que é o principal ponto em toda a estrutura.

A AC deve ter uma segurança muito maior do que a dos outros participantes da infra-estrutura porque se de alguma forma a chave privada da AC for violada, todo o sistema poderá estar comprometido. Essa chave privada poderia ser utilizada para assinar outros certificados gerados de forma ilícita. E esses falsos certificados tornaríam-se verdadeiros dentro da empresa.

Então, a política de segurança descrita para esse projeto deve ser implementada para o ótimo funcionamento da infra-estrutura.

Abrangência:

- Requisitos de segurança Física
- Requisitos de segurança Lógica

4.2.1 - Considerações Gerais

- Esta política deve ser comunicada para todo o pessoal envolvido e largamente divulgada através das entidades, garantindo que todos tenham consciência da mesma e a pratiquem dentro da empresa.
- Aconselha-se um programa de conscientização sobre segurança da informação para assegurar que todo o pessoal seja informado sobre os potenciais riscos de segurança e exposição a que estão submetidos os sistemas e operações das entidades.
- Previsão de mecanismo e repositório centralizado para ativação e manutenção de trilhas, logs e demais notificações de incidentes.

- Esta Política de Segurança deve ser revisada e atualizada periodicamente no máximo a cada 3 (três) anos, caso não ocorram eventos ou fatos relevantes que exijam uma revisão imediata.

4.2.2 - REQUISITOS DE SEGURANÇA DO AMBIENTE FÍSICO

4.2.2.1 - Responsabilidade da AC

- Sistemas de segurança para acesso físico como câmeras, portas com acesso restrito, etc, deverão ser instalados para controlar e auditar o acesso aos sistemas de certificação.
- Controles duplicados sobre o inventário e cartões/chaves de acesso deverão ser estabelecidos. Uma lista atualizada do pessoal que possui cartões/chaves deverá ser mantida.
- O servidor da AC deve ser mantido em área segura, protegido por um perímetro de segurança definido, com barreiras de segurança e controles de acesso. Deve ser fisicamente protegido de acesso não autorizado, dano, ou interferência. A proteção fornecida deve ser proporcional aos riscos identificados.
- A entrada e saída, nestas áreas ou partes dedicadas, deverão ser automaticamente registradas com data e hora definidas.
- Quaisquer equipamentos de gravação, fotografia, vídeo, som ou outro tipo de equipamento similar, não são permitidos dentro do ambiente do servidor da AC.
- O ambiente pertinente a AC deve ser monitorado, em tempo real, com as imagens registradas por meio de sistemas de CFTV.
- Sistemas de detecção de intrusos devem ser instalados e testados regularmente de forma a cobrir os ambientes, as portas e janelas acessíveis, no ambiente da AC.

4.2.3 - REQUISITOS DE SEGURANÇA DO AMBIENTE LÓGICO

4.2.3.1 - Responsabilidade da AC

- Os dados, as informações e os sistemas de informação das entidades e sob sua guarda devem ser protegidos contra ameaças e ações não autorizadas, acidentais ou não, de modo a reduzir riscos e garantir a integridade, sigilo e disponibilidade desses bens.
- As violações de segurança devem ser registradas e esses registros devem ser analisados periodicamente para os propósitos de caráter corretivo, legal e de auditoria.
- Os sistemas devem possuir controle de acesso de modo a assegurar o uso apenas a usuários ou processos autorizados. O responsável pela autorização ou confirmação da autorização deve ser claramente definido e registrado.
- Os arquivos de *logs* devem ser criteriosamente definidos para permitir recuperação nas situações de falhas, auditoria nas situações de violações de segurança e contabilização do uso de recursos. Os *logs* devem ser periodicamente analisados para identificar tendências, falhas ou usos indevidos.
- O acesso lógico, ao ambiente ou serviços disponíveis em servidores, deve ser controlado e protegido. As autorizações devem ser revistas, confirmadas e registradas continuadamente.
- O único acesso remoto ao servidor da AC disponível é para divulgação de certificados dos usuários e listas de certificados revogados.
- Devem ser adotados procedimentos sistematizados para monitorar a segurança do ambiente operacional, principalmente no que diz respeito à integridade dos arquivos de configuração do sistema operacional e de outros arquivos críticos.
- Proteção lógica adicional (criptografia) deve ser adotada para evitar o acesso não-autorizado às informações.
- A versão do sistema operacional, assim como outros *softwares* básicos instalados em máquinas servidoras, devem ser mantidos atualizados, em conformidade com as recomendações dos fabricantes.
- Os procedimentos de cópia de segurança (*backup*) e de recuperação devem estar documentados, mantidos atualizados e devem ser regularmente testados.
- Firewalls, IDS e outros tipos de proteção contra ataques externos devem ser empregados para garantir a segurança do servidor.

- As seguintes características das senhas de acesso devem estar definidas de forma adequada: conjunto de caracteres permitidos, tamanho mínimo e máximo, prazo de validade máximo, forma de troca e restrições específicas.
- O sistema de controle de acesso deve permitir ao usuário alterar sua senha sempre que desejar. A troca de uma senha bloqueada só deve ser executada após a identificação positiva do usuário. A senha digitada não deve ser exibida.
- O servidor deve sempre estar atualizado com antivírus e antitrojans para evitar que qualquer aplicativo dessa natureza comprometa a segurança.

4.2.3.2 - Responsabilidade do Cliente

- Devem ser adotadas medidas de segurança lógica referentes a combate a vírus, trojans, *backup*, controle de acesso e uso de software não autorizado.
- A entidade deverá estabelecer os aspectos de controle, distribuição e instalação de *softwares* utilizados.
- Os sistemas em uso devem solicitar nova autenticação após certo tempo de inatividade da sessão (*time-out*).
- Os procedimentos de combate a processos destrutivos (*vírus, cavalo-de-tróia e worms*) devem estar sistematizados e devem abranger máquinas servidoras, estações de trabalho, equipamentos portáteis e microcomputadores *stand alone* que utilizem a infra-estrutura.
- Os computadores pessoais devem ser protegidos por senhas de acesso ao sistema operacional e o acesso REMOTO aos computadores devem ser feitos apenas por funcionários ou serviços autorizados.

4.3 - Requisitos e Instalação

Existem procedimentos relativos à instalação e implantação do aplicativo desenvolvido neste projeto para gerenciamento e guarda das chaves criptográficas que se referem tanto a AC quanto ao computador pessoal utilizado pelos usuários finais.

Alguns pré-requisitos são exigidos para o bom funcionamento de toda a infra-estrutura. A instalação abrange desde a instalação do sistema operacional e configuração do mesmo até a instalação do aplicativo desenvolvido. Essa instalação deve ser seguida corretamente, enfatizando a parte relativa a segurança do terminal disponibilizado para servir como AC.

4.3.1 - Pré-Requisitos para a Autoridade Certificadora

O servidor para a AC deve conter a seguinte configuração física (Ou superior):

- Pentium III 1.0Ghz
- 512 Mb de memória Ram
- Placa de rede 10/100 Mbps
- Drive de disquete 1.44
- Monitor
- Teclado
- Mouse
- Porta USB

Softwares Necessários:

- Windows 2000 ou XP
- Java SDK 1.3
- Bibliotecas de criptografia da Bouncy Castle
- Apache Server 2.0 para Windows
- Um Firewall de escolha da empresa
- Internet Explorer
- Antivírus a escolha da empresa
- Antitrojans a escolha da empresa

- Pacote desenvolvido neste trabalho contendo o aplicativo para gerenciamento e guarda de chaves criptográficas.

4.3.2 - Instalação da AC

4.3.2.1 - Instalando o sistema operacional

- O Sistema operacional deve ser instalado em sua forma mais básica, sem disponibilidade de serviço adicional. Deverão ser ativados todos os serviços de auditoria relativos a gravação de logs ou trilhas.
- Deverão ser criados usuários para acesso ao sistema.
- O usuário administrador só deverá ser utilizado em emergências.
- A senha do usuário administrador deverá ser guardada em local seguro e com acesso restrito apenas a determinados funcionários que poderão fazer uso da mesma.
- Serviços do sistema operacional como Telnet, FTP ou qualquer outro tipo de acesso remoto diferente do oferecido pelo aplicativo APACHE não deverão estar habilitados.

4.3.2.2 - Instalando o JAVA SDK 1.3

- Instalar a versão 1.3 do JAVA SDK padrão. Não necessita de maiores alterações de segurança. A Virtual Machine já acompanha o pacote de instalação do Java e é necessária para o funcionamento do aplicativo relativo a AC.
- A instalação será necessária para a utilização do aplicativo.

Figura 16 - Instalação do Java

4.3.2.3 - Instalando o Apache Server 2

Nesse passo já é importante frizar a configuração do Servidor WEB. Um certo nível de segurança já é necessário nessa configuração e será apresentado nesse momento.

- Instalar o pacote do APACHE HTTP SERVER 2.0 ou superior para plataforma Windows.
- A instalação deve ser feita no modo padrão.
- O arquivo httpd.conf deve ser configurado conforme abaixo:

```
# Listen: Allows you to bind Apache to specific IP addresses and/or
# ports, instead of the default.
# Change this to Listen on specific IP addresses as shown below to
# prevent Apache from glomming onto all bound IP addresses (0.0.0.0)
#Listen 12.34.56.78:80
Listen 80
```

Nessa parte da configuração do servidor, o comando Listen deve estar setado apenas para responder na porta 80. Essa porta irá ser utilizada pelo aplicativo do usuário para conseguir informações sobre a CRL (Lista de Certificados Revogados).

Em seguida, segue o código de configuração do diretório reservado para a CRL. Esse diretório deve ser configurado para o local onde a lista for divulgada. Dessa forma, o usuário só poderá acessar esse diretório na forma de leitura.

```
# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this directory, but
# symbolic links and aliases may be used to point to other locations.
DocumentRoot "C:/CRL"
```

4.3.2.4 - Segurança lógica

A segurança implementada no servidor da AC é de extrema importância para a infra-estrutura. Esse tópico detalha um nível de segurança mínimo para que o servidor seja confiável.

- Instalar e configurar o Firewall escolhido. As configurações do firewall deverão ser definidas de forma a bloquear todo e qualquer pacote ou requisição que não seja do tipo TCP/IP na porta 80 do servidor.
- O antivírus escolhido deve ser instalado e a atualização do mesmo deve ser configurada como automática. A verificação de novas atualizações do antivírus deve ser diária.
- O antitrojan escolhido deve ser instalado e a atualização do mesmo deve ser configurada como automática. A verificação de novas atualizações do antitrojan deve ser diária.
- Os logs de acesso remoto e local devem ser gravados em disco e, se possível, utilizando algum tipo de criptografia simétrica.
- Configurar o servidor com um IP fixo na rede para responder as requisições de Lista de Certificados Revogados.
- Os usuários criados para acessarem o sistema operacional deverão ter acesso limitado às funções destinadas a eles.
- Mesmo com o firewall habilitado e configurado, os serviços de acesso remoto não permitidos deverão ser desativados.
- Deverá ser configurada uma proteção de tela com uma senha que se ativará em um tempo de 5 minutos.
- As atualizações automáticas do sistema operacional deverão estar habilitadas para que o próprio sistema se encarregue dessa função.

4.3.3 - Pré-Requisitos para o Cliente

O computador pessoal para o cliente deve conter a seguinte configuração física (Ou superior):

- Pentium III 500 Mhz
- 128 Mb de memória Ram
- Placa de rede 10/100 Mbps
- Drive de disquete 1.44
- Drive de CD-ROM
- Monitor
- Teclado
- Mouse
- Porta USB

Softwares Necessários:

- Windows 2000 ou XP
- Java Virtual Machine
- Internet Explorer
- Outlook Express
- Antivírus a escolha da empresa
- Antitrojans a escolha da empresa
- Pacote contendo o aplicativo para utilização e guarda de chaves criptográficas (Módulo do Cliente)

4.3.4 - Instalação do Cliente

4.3.4.1 - Instalando o sistema operacional

- O Sistema operacional deve ser instalado em modo padrão, sem disponibilidade de serviço adicional. Deverão ser ativados todos os serviços de auditoria relativos a gravação de logs ou trilhas.
- Os usuários de acesso a máquina ficam a critério da empresa.
- O usuário não terá acesso ao usuário administrador.
- Qualquer outro programa a ser instalado deverá estar de acordo com as normas da empresa.

4.3.4.2 - Segurança lógica

A segurança relativa ao cliente é importante para evitar fraudes em relação a sua chave privada. Entretanto, é importante frizar que essa chave é de responsabilidade do funcionário. Sua guarda, apesar de protegida pelo aplicativo, deve ser sempre observada pelo usuário.

- O antivírus escolhido deve ser instalado e a atualização do mesmo deve ser configurada como automática. A verificação de novas atualizações do antivírus deve ser diária.
- O anti-trojan escolhido deve ser instalado e a atualização do mesmo deve ser configurada como automática. A verificação de novas atualizações do antitrojan devem ser diárias.
- Os logs de acesso remoto e local devem ser gravados em disco e, se possível, utilizando algum tipo de criptografia simétrica.
- A estação de trabalho pode utilizar IP dinâmico.
- Os usuários com acesso ao terminal devem ter acessos limitados.
- Deverá ser configurada uma proteção de tela com senha que se ativará num tempo de 15 minutos.
- As atualizações automáticas do sistema operacional deverão estar habilitadas para que o próprio sistema se encarregue dessa função.

4.4 - Utilização da ICP projetada

Este tópico irá apresentar e explicar todo o escopo do aplicativo desenvolvido para o gerenciamento das chaves criptográficas em uma ICP. As imagens da aplicação estão sendo apresentadas de forma a facilitar a compreensão do mesmo. Através do aplicativo, será possível um melhor entendimento de uma infra-estrutura de chaves públicas, já que a função desta aplicação é a de exemplificar e embasar o projeto desenvolvido.

O projeto possui dois pólos distintos: o cliente e a AC.

4.4.1 – O módulo Cliente

O módulo definido na parte do cliente é mais complexo do que o da AC porque a interação entre os usuários é maior, mais freqüente e precisa ser muito mais elaborada do que a interface da AC, que irá apenas gerar certificados e listas de certificados revogados.

Este módulo define o uso da infra-estrutura de chaves públicas por parte do cliente. As funções são diversas. Conforme mostra a figura abaixo, as funcionalidades que serão apresentadas são: **Importação de certificados, Exportação do próprio certificado, Assinatura e Criptografia de Mensagens, Decifragem e verificação de assinaturas em mensagens e Requisição de Certificados para a AC.**

Figura 17 - Tela Principal do módulo Cliente

4.4.1.1 – Requisição de Certificados

Essa funcionalidade representa o início da participação de um usuário dentro de uma infra-estrutura de chaves públicas. O procedimento que aqui é executado irá gerar uma REQUISIÇÃO DE ASSINATURA DE CERTIFICADO (CSR) que deverá ser enviada para a AC. Neste momento é gerado o par de chaves que será utilizado na criptografia assimétrica. A chave privada ficará sob a guarda do usuário enquanto um protocolo no formato PKCS #10 irá ser gerado e enviado para a AC. O usuário deve lembrar que deverá estar presente na entrega desse arquivo, pois o certificado só será gerado mediante identificação do funcionário.

Figura 18 - Geração da CSR

Um funcionário responsável pela recepção da CSR irá comprovar se o usuário é ele mesmo através de uma documentação exigida, como o CPF e a Carteira de Identidade e em seguida irá impostar o nome e o CPF do usuário para poder gerar o certificado Digital.

4.4.1.2 – Importação de Certificados Digitais

Este processo representa a troca de certificados entre os usuários da ICP. Através desta funcionalidade do aplicativo, o usuário irá **receber** certificados digitais de outros usuários para poder trocar informações de forma segura e confiável. Os certificados que serão importados e definidos como confiáveis são públicos a qualquer participante da ICP dentro de uma corporação.

Neste momento ocorrem algumas verificações de grande importância para a segurança da infra-estrutura criada. Para se evitar que um indivíduo mal intencionado crie certificados de outros usuários, como o de um presidente, por exemplo, algumas verificações são necessárias.

Ao se importar um certificado, será feita uma verificação de cadeia de assinatura. Quando a AC gera um certificado, significa que aquele certificado foi assinado pela chave privada desta AC. Então, quando o aplicativo recebe um certificado, a chave pública da AC (conseguida no certificado da AC que também é público) é utilizada neste certificado recebido para se ter certeza de que foi realmente a autoridade certificadora em questão que assinou aquele certificado digital, tornando-o válido. Além disso, a aplicação irá buscar uma CRL (Lista de Certificados Revogados) junto a AC de forma a verificar se o certificado recebido não está revogado.

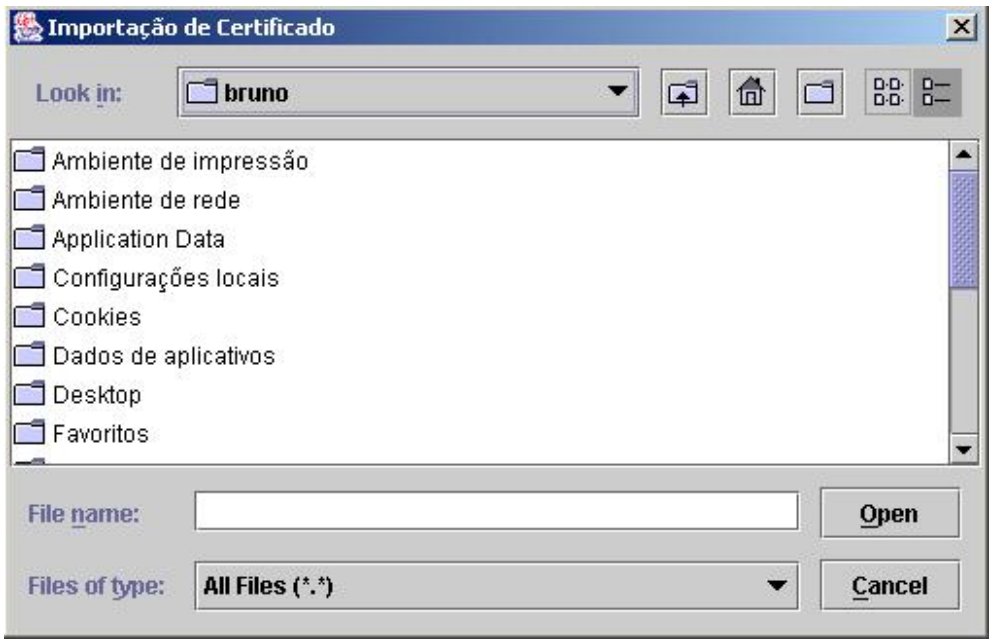

Figura 19 - Importação de Certificados

4.4.1.3 – Exportação do Certificado Digital

Figura 20 - Exportação de um certificado digital

O processo de exportação de um certificado digital é muito simples e faz parte da troca de certificados. O usuário poderá exportar o seu próprio certificado e o de outro usuário do serviço de ICP, caso deseje. Esse procedimento é transparente para o usuário e efetuado de forma segura, sem que haja qualquer interligação com a chave privada.

4.4.1.4 – Geração de uma Mensagem Assinada e Cifrada

A funcionalidade da geração de uma mensagem assinada e cifrada consiste no seguinte processo. O aplicativo recebe um texto digitado em sua interface com o cliente. Pode ser implementado também para atuar em um arquivo definido pelo usuário.

Em seguida, o procedimento que ele executa funciona de acordo com a seqüência abaixo:

1- Um algoritmo de HASH do tipo SHA1 é aplicado sobre a mensagem, gerando um resumo desta.
2- Nesse momento, a chave privada do usuário é utilizada para assinar o HASH que foi gerado anteriormente.
3- O certificado digital de um outro usuário é escolhido para se utilizar a chave pública do mesmo para a cifragem

Em seguida, o aplicativo irá salvar a informação assinada e cifrada em um arquivo com o padrão do protocolo PKCS #7 explicado anteriormente, sendo do tipo assinado e envelopado.

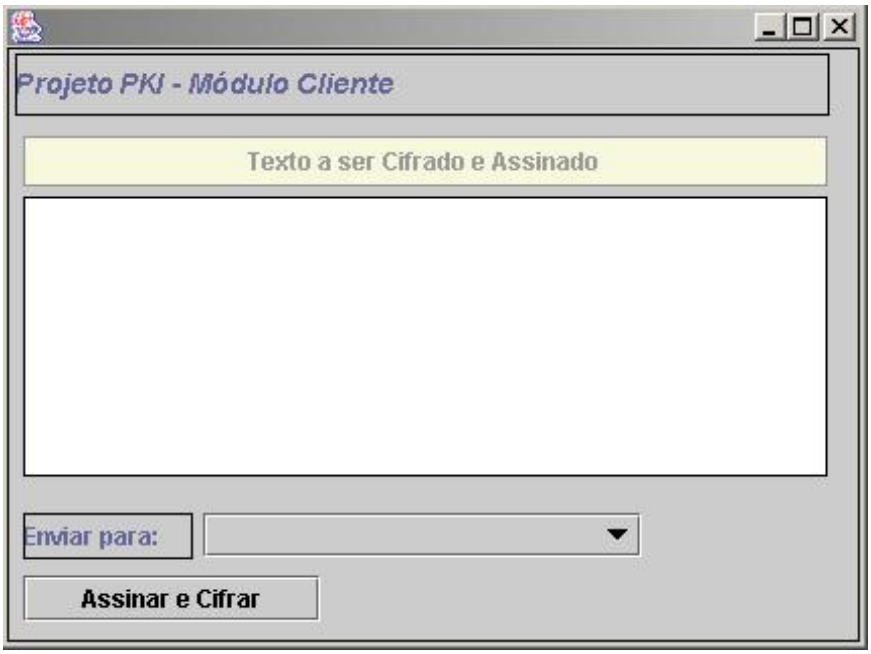

Figura 21 - Assinatura e cifragem de uma mensagem

4.4.1.5 – Recepção de mensagem assinada e cifrada

Figura 22 - Conferência e decifragem de uma mensagem

Esse procedimento é caracterizado pelo recebimento de uma mensagem assinada e cifrada por um outro usuário.

A aplicação então irá utilizar em primeiro lugar uma decifragem utilizando a chave privada do usuário que recebeu a mensagem. Após decifrar a mensagem, é necessário verificar a assinatura e a integridade dos dados.

O aplicativo irá utilizar a chave pública do usuário que enviou a mensagem e após isso irá gerar um HASH da mensagem que estava protegida no arquivo padrão PKCS #7. Se o HASH gerado sobre a mensagem for o mesmo que estava protegido pela chave privada do emissor da mensagem, então a informação será validada.

4.4.2 – O módulo AC

Este módulo define o uso da infra-estrutura de chaves públicas por parte da AC. As funções são divididas em dois processos: o de **geração de certificados** e o de **revogação de certificados**. Os usuários não possuem acesso a nenhum dos dois processos, sendo exclusivos apenas às pessoas com permissões para operar o terminal da Autoridade Certificadora.

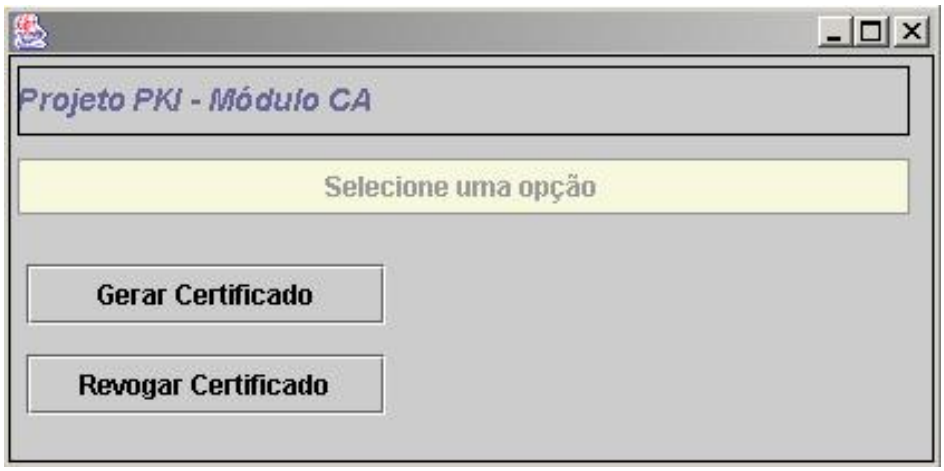

Figura 23 - Tela principal do módulo AC

A revogação dos certificados pode ser feita por qualquer razão que possa comprometer a infra-estrutura de chaves públicas.

Um usuário que pode ter tido sua chave privada roubada ou mesmo um usuário que esteja utilizando sua chave criptográfica transgredindo alguma norma da empresa pode ter seu certificado revogado, seja por solicitação ou por decisão da corporação.

4.4.2.1 – Geração de Certificados Digitais

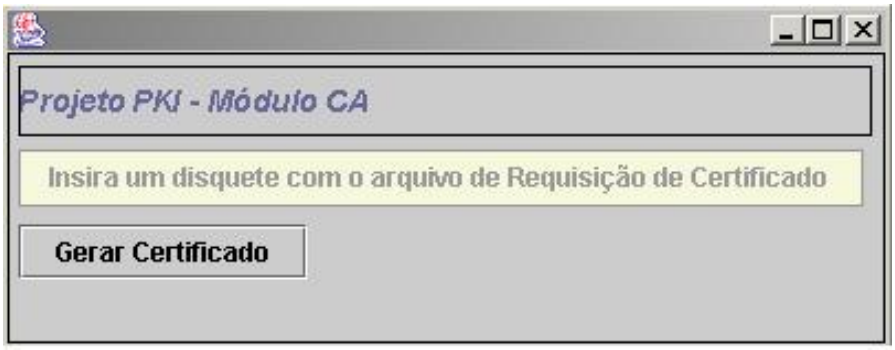

Figura 24 - Geração de certificados digitais

A geração de um certificado mediante solicitação ocorre neste processo. O cliente entrega um arquivo CSR para um operante da AC e recebe um certificado digital em seu nome.

4.4.2.2 – Revogação de Certificados

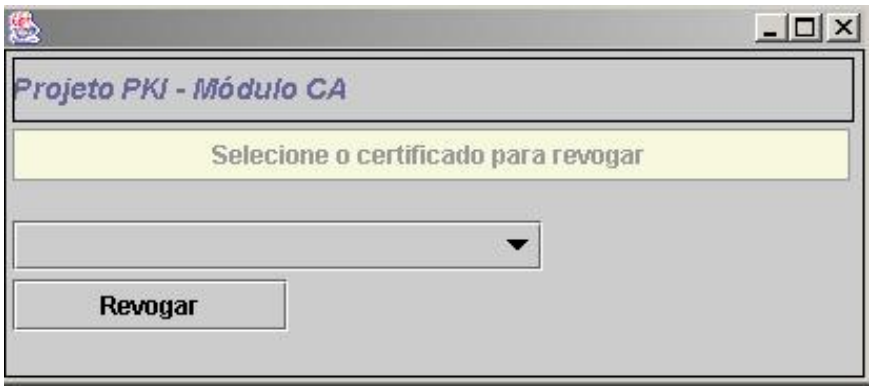

Figura 25 - Revogação de certificados digitais

O processo de revogação de certificados digitais é bem simples. Um usuário ou a corporação pode decidir revogar um certificado. Após ser revogado, o certificado passa a não ser mais válido dentro da infra-estrutura projetada. Essa revogação inclui o certificado cancelado na Lista de Certificados Revogados, que é utilizada pelo aplicativo para verificar se um certificado digital que será utilizado está ativo ou não.

4.5 - Decisões de Projeto

4.5.1 - Quanto à estrutura

A estrutura escolhida para o projeto foi do tipo hierárquica. Essa topologia foi selecionada porque em diversos aspectos se aproxima da formação da ICP-Brasil, que é a atual entidade raiz no Brasil.

Outros fatores também foram decisivos. A estrutura hierárquica, ao contrário da mista, é uma estrutura que demanda um menor tempo de implementação, já que a validação de cadeia de assinatura dos certificados é mais simples. A visualização da infra-estrutura é muito mais simples, já que se tem uma entidade raiz e todos os outros participantes dessa estrutura ficam situados abaixo.

É importante também enfatizar o lado da segurança dessa infra-estrutura. Qualquer problema que ocorra com a AC relativo a fraude poderá ser rapidamente contornado. Se a chave privada for corrompida, a AC poderá gerar uma outra chave rapidamente e distribuir a todos os participantes dependentes da sua infra-estrutura. Numa estrutura mista, é mais complexo, já que a distribuição do novo certificado vai além das dependências daquela AC.

A proposta do trabalho é implementar uma infra-estrutura interna a uma empresa, independente de outras Autoridades Certificadoras, sejam elas AC's do mercado ou particulares de outras empresas. Esse fator diminui o custo de implementação da infra-estrutura de chaves públicas. Os gastos e o tempo com implantação e desenvolvimento dos aplicativos são menores utilizando a topologia hierárquica.

4.5.2 - Quanto à política de segurança

A política de segurança foi definida da forma apresentada para garantir um mínimo de segurança a AC e aos clientes desta. Essa política de segurança pode ser incrementada, conforme a utilização da ICP ou o grau de importância das informações que trafegam pela rede.

Muitos aspectos foram herdados das normas da ICP-Brasil. Tanto as normas de segurança lógica e física foram estudadas para proporcionar uma boa segurança. As principais

características desta política de segurança implementada foram o tempo de implementação e o baixo custo.

Atualmente a implantação de uma AC no mercado exige um custo altíssimo em segurança física e lógica. E ainda traz mudanças consideráveis ao ambiente em que ela é instalada, como novas metodologias de trabalho e restrições de áreas.

A política deste trabalho não apresenta gastos altos para a empresa e muitos tópicos definidos, como o de segurança, já são inicialmente impostos pelas próprias empresas. Algumas normas já são implementadas pela corporação enquanto outras trazem baixos custos para implantação.

O ambiente físico restrito citado pode ser qualquer ambiente com um mínimo de segurança, onde não seja aberto a todos os funcionários. Isso já garante um nível de segurança. E a segurança lógica escolhida pode ser implementada pelo próprio sistema operacional com o auxílio de outros aplicativos como antivírus, antitrojans e firewalls, garantindo um segundo nível de segurança e limitando o número de usuários com permissão a acessar essa AC.

4.5.3 - Quanto ao desenvolvimento em linguagem JAVA

A linguagem Java foi utilizada para o desenvolvimento devido a suas características genéricas. A portabilidade desta linguagem permite que o aplicativo seja utilizado tanto em ambiente Windows quanto Linux. Mas o ambiente Linux possui algumas restrições, devido a apenas algumas novas versões possuírem suporte a certificados digitais.

Além da portabilidade, a linguagem oferece uma grande facilidade na implementação e, principalmente, uma segurança extra para o aplicativo.

A linguagem C poderia ter sido utilizada, juntamente com as bibliotecas de criptografia da Microsoft, mas isso limitaria a aplicação a um tipo de plataforma além de requerer um conhecimento muito amplo das bibliotecas da Microsoft. Isso demandaria um tempo desnecessário a implementação do projeto. Apesar da portabilidade, o aplicativo não foi testado no ambiente Linux e isso deverá ser feito futuramente.

O ambiente de desenvolvimento utilizado foi o Eclipse Platform, da IBM, que é uma ferramenta Free e auxilia na visualização do projeto e influencia no desenvolvimento do trabalho, além do próprio ambiente compilar e analisar os códigos escritos.

4.5.4 - Quanto às bibliotecas da Bouncy Castle

A organização Bouncy Castle trabalha especificamente com classes em Java desenvolvidas para manipulação de criptografia e certificação digital. Essa organização disponibiliza gratuitamente suas classes e a documentação, divulgando assim, seus serviços.

A utilização destas classes permitiu que a implementação do aplicativo fosse mais simples. Essas classes foram escolhidas para o desenvolvimento devido a grande quantidade de métodos que permitem ao programador trabalhar com geração de chaves simétricas ou assimétricas, geração de certificados digitais e até mesmo assinaturas digitais.

O escopo do projeto não abrange a implementação de algoritmos criptográficos, já que essa implementação é muito pesada e demandaria um estudo muito mais aprofundado na área matemática sobre criptografia. Então, optou-se por utilizar as classes da Bouncy Castle para utilização dos algoritmos mais comuns no mercado e para facilitar a geração e manipulação de certificados digitais.

Outros fatores da utilização destas classes foram a otimização destas classes, trabalhando de forma rápida com criptografia assimétrica e a compatibilidade com o padrão de certificados X.509, que está sendo utilizado no projeto.

4.5.5 - Quanto ao sistema operacional

O sistema operacional da Microsoft possui algumas características que eram necessárias à execução do projeto. A escolha do Windows 2000 para implementação do projeto foi embasada na facilidade com que esse sistema operacional possui de trabalhar com certificados digitais.

A Microsoft possui muitos recursos de acesso a certificados e outras facilidades que ainda não foram implementadas no sistema operacional Linux. Estes recursos foram decisivos para a escolha do sistema operacional a ser utilizado.

Além disso, são diversas as ferramentas de desenvolvimento visual e de código para a linguagem Java que são da plataforma Windows. Isso contribuiu com o andamento do trabalho.

4.5.6 - Quanto ao algoritmo RSA

Atualmente o algoritmo RSA é o mais utilizado no mercado e em diversas soluções que envolvem criptografia assimétrica. Sua segurança é indiscutivelmente muito forte, inibindo qualquer tipo de ataque que utilize força bruta. A escolha desse algoritmo foi embasado nestes dois fatores.

O projeto foi desenhado em cima desse algoritmo, pois o RSA possui uma grande aceitação no mercado, já que diversos tipos de testes e ataques já foram efetuados em cima desse algoritmo sem resultados.

O motivo da escolha baseado na força criptográfica deste algoritmo é simples. Esse algoritmo está há muito tempo no mercado e seu algoritmo é de domínio público. Isso comprova que o poder está na chave criptográfica, que torna inviável a quebra dessa criptografia. Então, acaba por se tornar um algoritmo muito seguro e confiável.

Não se sabe de nenhum ataque de força bruta que tenha tido sucesso em um curto espaço de tempo sobre o algoritmo RSA. E quanto maior o tamanho da chave, maior a dificuldade de se quebrar essa criptografia.

5 – Conclusão

Serviços como transações bancárias, comércios eletrônicos, entre outros, precisam de uma segurança concreta e uma garantia de ambos os participantes da transação. Através da criação de uma autoridade certificadora que valide os participantes, um aplicativo gerenciador de chaves criptográficas e uma política de segurança uma infra-estrutura foi construída para garantir essa troca de informações.

O projeto demonstra que atualmente a principal tecnologia para garantir processos básicos como confidencialidade, integridade e autenticidade no tráfego de uma rede de comunicação é o processo de certificação digital. Utilizando-se de criptografia assimétrica pode-se garantir esses processos.

A utilização pura e simples de pares de chaves para garantir requisitos como integridade, confidencialidade e autenticidade encontram seu grande obstáculo no gerenciamento destas chaves criptográficas.

O projeto apresentou uma solução para a garantia destes serviços de criptografia através de um modelo implementado que garante a geração e validação dos certificados, simulando uma autoridade certificadora que irá gerar e assinar todos os certificados digitais que estarão em uso dentro da infra-estrutura.

Diversos prejuízos podem ser evitados utilizando-se desta infra-estrutura criada, minimizando assim, as perdas de uma corporação.

Ficou também definido que apenas um aplicativo para tal gerenciamento não é suficiente. Uma política de segurança com uma boa base de informação ao usuário tornam-se fundamental para a utilização correta de uma ICP.

O trabalho abrangeu também a explicação de como funciona o processo de criptografia assimétrica e assinatura digital, que são conceitos importantes para todos os usuários da infra-estrutura.

Todo o objetivo do trabalho desenvolvido está focado para a segurança da informação que trafega em uma rede aberta, onde toda e qualquer informação pode ser interceptada ou alterada durante seu percurso. Diversas empresas já amargaram altos prejuízos por falhas de segurança quanto a confidencialidade de seus dados.

Avalizando o conceito de utilização do processo de Certificação Digital no Brasil, vem a ICP-Brasil regulamentando a infra-estrutura de chaves públicas brasileiras.

Finalizando, uma infra-estrutura de chaves públicas não pode ser implementada da noite para o dia confiando-se apenas na tecnologia utilizada. Deve-se ter a preocupação com

os usuários, principalmente. Estes devem ser informados sobre o funcionamento de uma ICP e conhecer a política de segurança nela definida. Dessa forma, será possível garantir integridade, confidencialidade e não-repúdio das informações digitais que trafegam dentro de um ambiente corporativo.

Implementações Futuras

Algumas alterações ou melhorias podem ser adicionadas ao projeto de forma a se ter uma infra-estrutura de chaves públicas mais segura e bem elaborada. São elas:

- Suporte a cartões criptográficos: A utilização de cartões de criptografia pode garantir uma maior segurança e maior flexibilidade do projeto. Entretanto, será necessário um maior estudo do sistema operacional a se utilizar, já que cada sistema possui sua própria comunicação com as leitoras de Smart Card.
- Implementação de uma estrutura mista: Com essa implementação, uma corporação pode trabalhar em parceria com outra, aceitando e validando certificados participantes de outras ICP's.
- Interação com aplicativos internos a empresa: Modularizar a aplicação de forma a se conseguir diversas funcionalidades separadas para qualquer serviço dentro de uma empresa possa utilizar os métodos de criptografia.
- Centralização dos diretórios: Os diretórios atualmente não estão centralizados. Cada usuário possui os seus diretórios de certificados. Implementando uma estrutura centralizada pode garantir maior segurança desta ICP.
- Melhoria da política de segurança do projeto: A política utilizada foi desenvolvida de forma básica, para não se gerar custos a corporação que pretende utilizar a ICP. Uma nova política pode ser desenvolvida e otimizada para se complementar a atual. Uma política de segurança bem elaborada pode ser decisiva em um projeto de ICP.

6 – Referências Bibliográficas

[1] - *Comércio & Segurança na WEB:* Simsom Garfinkel e Gene Sparfford – São Paulo, Market Press, 1999.

[2] - *Certificação Digital:* Fernando Augusto Higa, Jorge André R. N. Murr – monografia, Porto Alegre, 2002.

[3] - *Criptografia e Segurança - o guia oficial RSA:* Steve Burnett, Stephen Paine – Rio de Janeiro, Editora Campus, 2002.

[4] - *Segurança em Computação:* Ricardo Felipe Custódio, Santa Catarina, monografia, 2000.

[5] - *Infra-estrutura de chaves públicas – um estudo comparativo entre o modelo brasileiro e o modelo americano:* José Roberto Lenotti – monografia, Bauru 2002.

[6] - *RSA Criptografia Assimétrica e Assinatura Digital:* Luis Alberto M. Barbosa, Luis Fernando B., Marcelo L. Brisqui, Sirlei Cristina da Silva, Campinas, 2003.

[7] - *Plannig for PKI:* Russ Housley, Tim Polk – Canada, Wiley Computer Publishing, 2001.

[8] - *RFC [3029]* - Internet X.509 Public Key Infrastructure Data Validation and Certification Server Protocols, 2001.

[9] - *Introduction to the PKCS Standards:* Mohan Atreya, tutorial.

[10] - *PKCS #10 v1.7 - Certification Request Syntax Standard:* RSA Laboratories, *2000*.

[11] - *RFC [2315] - PKCS #7:* Cryptographic Message Syntax Version 1.5, B. Kaliski, 1998.

[12] - *Instituto Nacional de Tecnologia da Informação:* www.iti.gov.br

[13] - *Política de Segurança da ICP-Brasil (*Aprovada pela Resolução n. 2, de 25 de Setembro de 2001, do comitê gestor da ICP-Brasil).

[14] - *RSA Security:* www.rsasecurity.com

7 – Bibliografia

Computer Security Journal: Carl Ellison, Bruce Schneier – Volume XVI, Number 1, 2000.

Internet X.509 Public Key Infrastructure: S. Boyen, Hallam-Baker, 2000.

PKI Architetures and Interoperability: W.E. Burr. 1998.

Eletronic Commerce - A Managerial Perspective: TURBAN, Efraim; LEE, Jae; KING, David; CHUNG, H. Michael. Prentice-Hall Inc. New Jersey. 2000.

Modelo, Aspectos e Contribuições de sua Aplicação: ALBERTIN, Luiz Alberto - Editora Atlas S.A., São Paulo, 2001.

A Method for Obtaining Digital Signatures and Public-Key Cryptosystems: RIVEST R.L., SHAMIR A., ADLEMAN L.M, Communications of the ACM, 1978.

Comitê Gestor Da Icp-Brasil: Resolução No 7, De 12 De Dezembro De 2001.

Telecommunication Standardization Sector Of Itu: Recommendation X.509, 1995.

Java – Como programar: Harvey M. Deitel, Paul J. Deitel, Editora Bookman, 4ª edição, 2002.

Trust Models and Management in Public-Key Infrastructures: John Linn, RSA Laboratories, 2000.

Understanding PKI - Concepts, Standards, and Deployment Considerations Edition: Carlisle Adams, Steve Lloyd, Addison-Wesley Pub, 2002.

Manual de Segurança do SPB: L. Germano, Guimarães, 2002.

Módulo Security: www.modulo.com.br

Bouncy Castle: www.bouncycastle.org/documentation.html

8 – Anexos

ANEXO A – Código fonte do módulo Autoridade Certificadora

Main.java

```
package ca;

import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import ca.controle.ControleCA;
import ca.ui.Principal;


/**
 * Autor: Bruno
 *
 * Chamada principal da tela do Modulo Cliente
 */
public class Main {


	public Main() {
		super();
	}

	public static void main(String[] args) {

		final Principal frame = new Principal();
		frame.addWindowListener(
			new WindowAdapter(){
				public void windowClosing(WindowEvent we){
					frame.dispose();
					System.exit(0);
				}
			}
		);
		ControleCA controle = new ControleCA(frame);
		controle.iniciar();
	}

}
```

ControleCA.java

```
/**
 * Autor: Bruno
 *
 * Classe responsável por gerenciar o load da parte da Autoridade
Certificadora
 *
 * Gera diretórios de certificados validos e revogados.
 * Verifica se já existe um certificado gerado para a AC.
 * Caso não exista, gera um auto-assinado.
 *
 */
package ca.controle;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.security.KeyPair;
import java.security.PrivateKey;

import javax.swing.JOptionPane;

import org.bouncycastle.jce.provider.X509CertificateObject;

import pki.Utils;

import ca.ui.GeraCert;
import ca.ui.Principal;
import ca.ui.RevogarCert;


public class ControleCA implements ActionListener {

	/* constantes do sistema */
	public static final String DIRETORIO_CERTIFICADOS =
	"certificadosgerados";
	public static final String DIRETORIO_CERTIFICADOS_REVOGADOS = "c:" +
	File.separator + "revogados" + File.separator;

	public static final String ARQUIVO_CERTIFICADO_CA = "ca.crt";
	public static final String ARQUIVO_CHAVE_CA = "ca.p12";

	public static final String SUBJECT_CA = "CN=CA TESTE, O=Autoridade
	Certificadora de Teste, OU=Seguranca da Informacao, ST=DF, C=BR";
	public static final String SENHA_CA = "1234";
	public static final String ALIAS_CA = "ca-teste";

	private Principal frame;


	public ControleCA(Principal frame) {
		super();

		this.frame = frame;
		this.frame.getJButton().addActionListener(this);
		this.frame.getJButton1().addActionListener(this);
	}
```

```
public void iniciar(){

	//cria diretorios se necessario
	new File(DIRETORIO_CERTIFICADOS).mkdirs();
	new File(DIRETORIO_CERTIFICADOS_REVOGADOS).mkdirs();
	//gera certificado autoassinado da CA se necessario
	if(!new File(ARQUIVO_CHAVE_CA).exists() || !new
	File(ARQUIVO_CERTIFICADO_CA).exists()){

		//avisa a criacao do certificado da CA
		JOptionPane.showMessageDialog(frame, "Será gerado agora um
		certificado raíz para a Autoridade Certificadora (CA)");

		try{
			//gera o par de chaves
			KeyPair kp = Utils.gerarKeyPairRSA();

			//gera um certificado auto assinado, logo o
			proprietario(subject) e o emissor(issuer) são iguais
			X509CertificateObject cert =
			Utils.gerarCertificadoX509V3(SUBJECT_CA, SUBJECT_CA,
			"01", kp, "SHA1withRSAEncryption");

			//grava a chave privada em PKCS 12 e o certificado em um
			outro arquivo
			Utils.gravarPrivKeyPkcs12(ARQUIVO_CHAVE_CA,
			SENHA_CA.toCharArray(), kp.getPrivate(), ALIAS_CA, cert);
			Utils.gravarCertificado(cert, ARQUIVO_CERTIFICADO_CA);

			//ok
			JOptionPane.showMessageDialog(frame, "O certificado raíz
			para a Autoridade Certificadora foi gerado com
			sucesso!");


		}catch(Exception e){
			e.printStackTrace();
			JOptionPane.showMessageDialog(frame, "Erro ao gerar o
			certificado raíz da CA: "+e.getMessage());
		}
	}

	frame.show();
}
	public void actionPerformed(ActionEvent e) {

		if(e.getSource() == this.frame.getJButton()){

			//Gerar Certificado
			new GeraCert().show();

		}else if(e.getSource() == this.frame.getJButton1()){
			//Revogar Certificado
			new RevogarCert().show();
		}

	}

}
```

Principal.Java

```
/**
 * Autor: Bruno
 *
 * Tela principal do aplicativo da CA com suas funcionalidades.
 *
 */

package ca.ui;
import javax.swing.JFrame;

public class Principal extends JFrame {

	private javax.swing.JPanel jContentPane = null;

	private javax.swing.JLabel jLabel = null;
	private javax.swing.JTextField jTextField = null;
	private javax.swing.JButton jButton = null;
	private javax.swing.JButton jButton1 = null;

	public Principal() {
		super();
		initialize();
	}

	private void initialize() {
		this.setSize(423, 208);
		this.setContentPane(getJContentPane());
		this.setTitle("Módulo CA");
	}


	private javax.swing.JPanel getJContentPane() {
		if (jContentPane == null) {
			jContentPane = new javax.swing.JPanel();
			jContentPane.setLayout(null);
			jContentPane.add(getJLabel(), null);
			jContentPane.add(getJTextField(), null);
			jContentPane.add(getJButton(), null);
			jContentPane.add(getJButton1(), null);
		}
		return jContentPane;
	}


	private javax.swing.JLabel getJLabel() {
		if(jLabel == null) {
			jLabel = new javax.swing.JLabel();
			jLabel.setBounds(4, 5, 400, 31);
			jLabel.setText("Projeto PKI - Módulo CA");
			jLabel.setFont(new java.awt.Font("Dialog",
			java.awt.Font.BOLD | java.awt.Font.ITALIC, 14));
		}
		return jLabel;
	}


	private javax.swing.JTextField getJTextField() {
		if(jTextField == null) {
```

```
                jTextField = new javax.swing.JTextField();
                jTextField.setBounds(4, 46, 400, 25);
                jTextField.setBackground(new
                java.awt.Color(247,248,222));
                jTextField.setText("Selecione uma opção");
                jTextField.setEnabled(false);
                jTextField.setFont(new java.awt.Font("Dialog",
                java.awt.Font.BOLD, 12));

        jTextField.setHorizontalAlignment(javax.swing.JTextField.CENTER);
            }
            return jTextField;
        }


        public javax.swing.JButton getJButton() {
            if(jButton == null) {
                jButton = new javax.swing.JButton();
                jButton.setBounds(8, 93, 161, 27);
                jButton.setText("Gerar Certificado");
            }
            return jButton;
        }


        public javax.swing.JButton getJButton1() {
            if(jButton1 == null) {
                jButton1 = new javax.swing.JButton();
                jButton1.setBounds(8, 133, 161, 27);
                jButton1.setText("Revogar Certificado");
            }
            return jButton1;
        }
}
```

GeraCert.Java

```
/**
 * Autor: Bruno
 *
 * Classe que faz a geração de um certificado a partir
 * de um CSR fornecido pelo cliente.
 *
 */
package ca.ui;

import java.io.File;
import java.math.BigInteger;
import java.security.KeyPair;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.SecureRandom;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import org.bouncycastle.jce.PKCS10CertificationRequest;
import org.bouncycastle.jce.provider.X509CertificateObject;
import ca.controle.ControleCA;
import pki.Utils;
import util.FileUtils;
import util.GuiUtils;


public class GeraCert extends JFrame {

	private javax.swing.JPanel jContentPane = null;

	private javax.swing.JLabel jLabel = null;
	private javax.swing.JTextField jTextField = null;
	private javax.swing.JButton jButton = null;


	public GeraCert() {
		super();
		initialize();
	}


	private void initialize() {
		this.setSize(406, 159);
		this.setContentPane(getJContentPane());
	}


	private javax.swing.JPanel getJContentPane() {
		if (jContentPane == null) {
			jContentPane = new javax.swing.JPanel();
			jContentPane.setLayout(null);
			jContentPane.add(getJLabel(), null);
			jContentPane.add(getJTextField(), null);
			jContentPane.add(getJButton(), null);
		}
		return jContentPane;
	}
```

```
	private javax.swing.JLabel getJLabel() {
		if(jLabel == null) {
			jLabel = new javax.swing.JLabel();
			jLabel.setBounds(5, 6, 387, 32);
			jLabel.setText("Projeto PKI - Módulo CA");
			jLabel.setFont(new java.awt.Font("Dialog",
			java.awt.Font.BOLD | java.awt.Font.ITALIC, 14));
		}
		return jLabel;
	}


	private javax.swing.JTextField getJTextField() {
		if(jTextField == null) {
			jTextField = new javax.swing.JTextField();
			jTextField.setBounds(5, 44, 383, 26);
			jTextField.setBackground(new
			java.awt.Color(247,248,222));
			jTextField.setText("Insira um disquete com o arquivo de
Requisição de Certificado");
			jTextField.setEnabled(false);
			jTextField.setFont(new java.awt.Font("Dialog",
			java.awt.Font.BOLD, 12));
			jTextField.setHorizontalAlignment(javax.swing.JTextField.
			CENTER);
		}
		return jTextField;
	}


	private javax.swing.JButton getJButton() {
		if(jButton == null) {
			jButton = new javax.swing.JButton();
			jButton.setBounds(5, 78, 131, 25);
			jButton.setText("Gerar Certificado");
			jButton.addActionListener(new
			java.awt.event.ActionListener() {
			public void actionPerformed(java.awt.event.ActionEvent e)
			{
				gerarCertificado();
			}
		});
	  }
		return jButton;
	}


	 /**
	 * Gera certificado a partir de uma requisicao.
	 */
	private void gerarCertificado(){

		try{

			//pergunta por um arquivo ao usuario
			File arq = GuiUtils.promptFileForOpen(this, "Abrir
			Requisição de Certificado", null);
			if(arq == null){
				//usuario abortou
				return;
			}
```

```
//tenta ler a requisicao de certificado
PKCS10CertificationRequest req =
Utils.getInstanceRequisicaoCertificado(arq.getAbsolutePath());

//extrai o proprietario
String proprietario =
req.getCertificationRequestInfo().getSubject().toString();

//recupera a chave publica da requisicao
PublicKey pubKey = req.getPublicKey();

//le a chave privada da ca
PrivateKey privKey =
Utils.lerPrivKeyPkcs12(ControleCA.ARQUIVO_CHAVE_CA,
ControleCA.SENHA_CA.toCharArray(), ControleCA.ALIAS_CA);

//le o certificado da ca
X509CertificateObject certCA =
Utils.getInstanceCertificadoX509(ControleCA.ARQUIVO_CERTIFICADO_CA);

//extrai o emissor
String emissor = certCA.getSubjectDN().getName();

//gera numero serial
int qtdeArquivos = (new
File(ControleCA.DIRETORIO_CERTIFICADOS).list().length) + 1;
String serialNumber = Integer.toString(qtdeArquivos);

//insere a chave publica do proprietario e assina o certificado com a
chave privada da CA
KeyPair kp = new KeyPair(pubKey, privKey);

//gera certificado assinado pela CA
X509CertificateObject cert =
Utils.gerarCertificadoX509V3(proprietario, emissor, serialNumber, kp,
"SHA1withRSA");


//grava arquivo no diretorio da CA
String nmArqCertNovo = ControleCA.DIRETORIO_CERTIFICADOS +
File.separator + serialNumber+".crt";
Utils.gravarCertificado(cert, nmArqCertNovo);


//pergunta por um arquivo para salvar o certificado
File arqCertCliente = GuiUtils.promptFileForSave(this, "Salvar
Certificado", null);

if(arqCertCliente == null){
      //usuario abortou
      return;
}

//grava o certificado
FileUtils.copiarArquivo(new File(nmArqCertNovo), arqCertCliente);

//grava uma copia do root tambem
File rootCer = new File( arqCertCliente.getParentFile(), "root-
"+arqCertCliente.getName());
```

```
        FileUtils.copiarArquivo(new File(ControleCA.ARQUIVO_CERTIFICADO_CA),
        rootCer);


        //ok
        JOptionPane.showMessageDialog(this, "Certificado gerado com
        sucesso!");


        }catch(Exception e){
              e.printStackTrace();
              JOptionPane.showMessageDialog(this, "Erro ao gerar certificado:
              "+e.getMessage());
        }
    }

}
```

RevogarCert.Java

```
/**
 * Autor: Bruno
 *
 * Classe que faz a revogação dos certificados selecionados.
 *
 */
package ca.ui;

import java.io.File;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import org.bouncycastle.jce.provider.X509CertificateObject;
import pki.Utils;
import ca.controle.ControleCA;
import util.FileUtils;

public class RevogarCert extends JFrame {

	private javax.swing.JPanel jContentPane = null;

	private javax.swing.JLabel jLabel = null;
	private javax.swing.JTextField jTextField = null;
	private javax.swing.JComboBox jComboBox = null;
	private javax.swing.JButton jButton = null;


	public RevogarCert() {
		super();
		initialize();
	}


	private void initialize() {
		this.setSize(417, 184);
		this.setContentPane(getJContentPane());

		//exibe a lista de certificados no combo
		listarCertificados();
	}



	private void listarCertificados(){

		//lista os certificados
		File dir = new File(ControleCA.DIRETORIO_CERTIFICADOS);

		File arqs[] = dir.listFiles();
		String nmArq;
		String proprietario;
		X509CertificateObject cert;

		for (int i = 0; i < arqs.length; i++) {

			//nome do arquivo
			nmArq = arqs[i].getName();

			//testa a extensao do arquivo
```

```
            if(nmArq.indexOf(".crt") > 0){

                  try{
                        //recupera o proprietario do certificado
                        cert =
                        Utils.getInstanceCertificadoX509(arqs[i].getAb
                        solutePath());
                        proprietario = cert.getSubjectDN().getName();
                        proprietario = Utils.extrairCN(proprietario);

                        getJComboBox().addItem(nmArq + " - " +
                        proprietario);

                  }catch (Exception e) {
                        //apenas nao adiciona se falhou
                  }
            }
      }



}



private javax.swing.JPanel getJContentPane() {
      if (jContentPane == null) {
            jContentPane = new javax.swing.JPanel();
            jContentPane.setLayout(null);
            jContentPane.add(getJLabel(), null);
            jContentPane.add(getJTextField(), null);
            jContentPane.add(getJComboBox(), null);
            jContentPane.add(getJButton(), null);
      }
      return jContentPane;
}


private javax.swing.JLabel getJLabel() {
      if(jLabel == null) {
            jLabel = new javax.swing.JLabel();
            jLabel.setBounds(4, 4, 400, 31);
            jLabel.setText("Projeto PKI - Módulo CA");
            jLabel.setFont(new java.awt.Font("Dialog",
            java.awt.Font.BOLD | java.awt.Font.ITALIC, 14));
      }
      return jLabel;
}


private javax.swing.JTextField getJTextField() {
      if(jTextField == null) {
            jTextField = new javax.swing.JTextField();
            jTextField.setBounds(4, 38, 398, 25);
            jTextField.setBackground(new
            java.awt.Color(247,248,222));
            jTextField.setText("Selecione o certificado para
            revogar");
            jTextField.setEnabled(false);
            jTextField.setFont(new java.awt.Font("Dialog",
            java.awt.Font.BOLD, 12));
```

```
		jTextField.setHorizontalAlignment(javax.swing.JTextField.CENTER);
		}
		return jTextField;
	}



	private javax.swing.JComboBox getJComboBox() {
		if(jComboBox == null) {
			jComboBox = new javax.swing.JComboBox();
			jComboBox.setBounds(4, 82, 252, 24);
		}
		return jComboBox;
	}



	private javax.swing.JButton getJButton() {
		if(jButton == null) {
			jButton = new javax.swing.JButton();
			jButton.setBounds(4, 110, 131, 25);
			jButton.setText("Revogar");
			jButton.addActionListener(new
			java.awt.event.ActionListener() {
			public void actionPerformed(java.awt.event.ActionEvent e)
{
					revogarCertificado();
				}
			});
		}
		return jButton;
	}

	/**
	 *
	 * Revoga o certificado selecionado.
	 *
	 */
	private void revogarCertificado(){

		String selecao = (String) getJComboBox().getSelectedItem();

		if(selecao != null){

			//recupera o nome do arquivo com a extensao
			int i = selecao.indexOf(".crt");
			String nmArq = selecao.substring(0, i + 4);
			String nmArqCer = ControleCA.DIRETORIO_CERTIFICADOS +
			File.separator + nmArq;

			//copia para o diretorio de certificados revogados
			String nmArqCerRevogado =
			ControleCA.DIRETORIO_CERTIFICADOS_REVOGADOS +
			File.separator + nmArq;
			try{

				FileUtils.copiarArquivo(new File(nmArqCer), new
				File(nmArqCerRevogado));

				//ok
```

```
                        JOptionPane.showMessageDialog(this, "Revogação de
                        certificado realizada com sucesso!");

                  }catch(Exception e){
                        e.printStackTrace();
                        JOptionPane.showMessageDialog(this, "Erro ao
                        revogar certificado: "+e.getMessage());
                  }
            }

      }

}
```

ANEXO B – Código fonte do módulo Cliente

Main.Java

```
/**
 * Autor: Bruno
 *
 * Classe principal para chamada da tela
 * inicial do módulo cliente
 */
package cliente;

import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import cliente.controle.ControleCliente;
import cliente.ui.Principal;

public class Main {



	public Main() {
		super();
	}

	public static void main(String[] args) {

		final Principal frame = new Principal();
		frame.addWindowListener(
			new WindowAdapter(){
				public void windowClosing(WindowEvent we){
					frame.dispose();
					System.exit(0);
				}
			}
		);
		ControleCliente controle = new ControleCliente(frame);
		controle.iniciar();

	}
}
```

ControleCliente.Java

```
/**
 * Autor: Bruno
 *
 * Classe responsável por gerenciar o load da parte do cliente
 *
 * Gera diretórios de certificados confiáveis e revogados.
 * Verifica se já existe um certificado gerado para o cliente.
 * Verifica se existe uma CA confiável e se existe uma chave
 * privada para o funcionário em questão.
 *
 */
package cliente.controle;

import java.awt.Component;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.cert.X509Certificate;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import javax.swing.JOptionPane;
import org.bouncycastle.jce.provider.X509CertificateObject;
import pki.Utils;
import util.FileUtils;
import util.GuiUtils;
import cliente.ui.EnviaMsg;
import cliente.ui.Exportacao;
import cliente.ui.GeraCSR;
import cliente.ui.Importacao;
import cliente.ui.Principal;
import cliente.ui.ReceberMsg;


public class ControleCliente implements ActionListener {


	/* constantes do sistema */
	public static final String DIRETORIO_CERTIFICADOS = "certificados";
	public static final String DIRETORIO_CERTIFICADOS_REVOGADOS = "c:" +
	File.separator + "revogados" + File.separator;

	public static final String ARQUIVO_CERTIFICADO_CLIENTE =
	"cliente.crt";
	public static final String ARQUIVO_CERTIFICADO_ROOT = "root.crt";
	public static final String ARQUIVO_CHAVE_TEMP_CLIENTE = "cliente.p5";
	public static final String ARQUIVO_CHAVE_CLIENTE = "cliente.p12";

	public static final String SUBJECT_CLIENTE = "O=Empresa Teste,
	OU=CPD, ST=DF, C=BR, CN=";
	public static final String SENHA_CLIENTE = "1234";
	public static final String ALIAS_CLIENTE = "cliente-teste";


	private Principal frame;
```

```
public ControleCliente(Principal frame) {
	super();

	this.frame = frame;
	this.frame.getJButton().addActionListener(this);
	this.frame.getJButton1().addActionListener(this);
	this.frame.getJButton2().addActionListener(this);
	this.frame.getJButton3().addActionListener(this);
	this.frame.getJButton4().addActionListener(this);
}

public void iniciar(){

	//cria diretorios se necessario
	new File(DIRETORIO_CERTIFICADOS).mkdirs();
	new File(DIRETORIO_CERTIFICADOS_REVOGADOS).mkdirs();

	//verifica se jah existe certificado instalado
	if(!new File(ARQUIVO_CHAVE_CLIENTE).exists() || !new
	File(ARQUIVO_CERTIFICADO_CLIENTE).exists()){

	//avisa a que deve ser criado um certificado para o cliente

	JOptionPane.showMessageDialog(frame, "Nenhum certificado foi
	criado para este cliente,\n use a opção \"Requisição de
	Certificado\" para gerar um\n e depois a opção \"Importar
	Certificado\" para cadastrá-lo.");
	}else{

		try{
			//verifica se o certificado do usuario foi revogado
			X509Certificate certUsuario =
			Utils.getInstanceCertificadoX509(ARQUIVO_CERTIFICAD
			O_CLIENTE);
			if(ControleCliente.isCertificadoRevogado(frame,
			certUsuario)){
				JOptionPane.showMessageDialog(frame, "O
				certificado do usuário foi revogado, favor
				requisitar um novo certificado.");
			}
		}catch(Exception e){
			e.printStackTrace();
		}

	}


	frame.show();
}



public void actionPerformed(ActionEvent e) {

	if(e.getSource() == this.frame.getJButton()){

		//precisa que o root jah esteja cadastrado

		if(!new File(ARQUIVO_CERTIFICADO_ROOT).exists()){
			//cancela a importacao
```

```
                        JOptionPane.showMessageDialog(frame, "Importe
                        primeiro o certificado de uma Autoridade
                        Certificadora confiável!");
                        return;
                  }

                  //Assinar e Criptografar Mensagens
                  new EnviaMsg().show();

            }else if(e.getSource() == this.frame.getJButton1()){

                  //Importar Certificado
                  importarCertificado();
                  //new Importacao().show();

            }else if(e.getSource() == this.frame.getJButton2()){

                  //precisa que o root jah esteja cadastrado

                  if(!new File(ARQUIVO_CERTIFICADO_ROOT).exists()){
                        //cancela a importacao
                        JOptionPane.showMessageDialog(frame, "Importe
                        primeiro o certificado de uma Autoridade
                        Certificadora confiável!");
                        return;
                  }

                  //Exportar Certificado
                  new Exportacao().show();

            }else if(e.getSource() == this.frame.getJButton3()){

                  //precisa que o root jah esteja cadastrado

                  if(!new File(ARQUIVO_CERTIFICADO_ROOT).exists()){
                        //cancela a importacao
                        JOptionPane.showMessageDialog(frame, "Importe
                        primeiro o certificado de uma Autoridade
                        Certificadora confiável!");
                        return;
                  }

                  //Receber e Verificar Assinatura na Mensagem
                  new ReceberMsg().show();

            }else if(e.getSource() == this.frame.getJButton4()){

                  //Requisição de Certificado
                  new GeraCSR().show();
            }

      }

      private void importarCertificado(){

            //pergunta ao usuario pelo certificado
            File arq = GuiUtils.promptFileForOpen(frame, "Importação de
            Certificado", null);
            if(arq == null){
                  //abortado pelo usuario
                  return;
```

```
}

try{

	//tenta abrir o arquivo
	X509CertificateObject cert =
	Utils.getInstanceCertificadoX509(arq.getAbsolutePath());

	//tenta ler a chave privada temporaria
	File arqChaveTemp = new File(ARQUIVO_CHAVE_TEMP_CLIENTE);

	//verifica se eh um certificado de AC
	if(cert.getIssuerDN().equals(cert.getSubjectDN())){

		//arquivo de root AC
		FileUtils.copiarArquivo(arq, new
		File(ARQUIVO_CERTIFICADO_ROOT));

		JOptionPane.showMessageDialog(frame, "Importação do
		certificado de AC foi realizada com sucesso!");
		return;

	}else {

		//valida AC
		if(!validarRoot(frame, cert)){
			return;
		}

		//verifica se existe a chave temporaria
		if(arqChaveTemp.exists()){
			//chave publica do certificado a importar
			RSAPublicKey pubKey = (RSAPublicKey)
			cert.getPublicKey();

			//verifica se o certificado corresponde com a
			chave privada
			RSAPrivateKey privKey = (RSAPrivateKey)
			Utils.lerPrivKeyPkcs5(ARQUIVO_CHAVE_TEMP_CLIEN
			TE, SENHA_CLIENTE.toCharArray());
			if(Utils.isChavePublicaConfere(privKey,
			pubKey)){

			//copia para o certificado pessoal

			FileUtils.copiarArquivo(arq, new
			File(ARQUIVO_CERTIFICADO_CLIENTE));

			//cria PKCS12 para armazenar a chave privada
			Utils.gravarPrivKeyPkcs12(ARQUIVO_CHAVE_CLIENT
			E, SENHA_CLIENTE.toCharArray(), privKey,
			ALIAS_CLIENTE, cert);

			//apaga a chave privada temporaria
			arqChaveTemp.delete();

			//ok
			JOptionPane.showMessageDialog(frame,
			"Importação do certificado pessoal foi
			realizada com sucesso!");
			return;
```

```
                }
            }

        //copia para o diretorio de certificados
        int qtdeArquivos = (new
        File(ControleCliente.DIRETORIO_CERTIFICADOS).list().length) +
        1;
        String nmArqCertNovo = DIRETORIO_CERTIFICADOS + File.separator
        + Integer.toString(qtdeArquivos) + ".crt";
        FileUtils.copiarArquivo(arq, new File(nmArqCertNovo));

        //ok
        JOptionPane.showMessageDialog(frame, "Importação de certificado
        realizada com sucesso!");
            }


        }catch(Exception e){
            e.printStackTrace();
            JOptionPane.showMessageDialog(frame, "Erro ao importar
            certificado: "+e.getMessage());
        }
}


/**
 * Valida se um certificado pertence a AC cadastrada.
 *
 */
public static boolean validarRoot(Component frame, X509Certificate
cert){
        //valida AC
        try{

            if(!new File(ARQUIVO_CERTIFICADO_ROOT).exists()){
                //cancela a importacao
                JOptionPane.showMessageDialog(frame, "Importe
                primeiro o certificado de uma Autoridade
                Certificadora confiável!");

            }else{
                X509CertificateObject certRoot =
                Utils.getInstanceCertificadoX509(ARQUIVO_CERTIFICAD
                O_ROOT);
                cert.verify(certRoot.getPublicKey());

                return true;
            }


        }catch(Exception ex){

            //cancela a importacao
            JOptionPane.showMessageDialog(frame, "Erro: Este
            certificado não pertence a uma Autoridade Certificadora
            confiável!");
        }

        return false;
}
```

```
	/**
	 * Valida se um certificado pertence a AC cadastrada.
	 *
	 */
	public static boolean isCertificadoRevogado(Component frame,
	X509Certificate certTeste){


	//lista os certificados
	File dir = new
	File(ControleCliente.DIRETORIO_CERTIFICADOS_REVOGADOS);

	File arqs[] = dir.listFiles();
	String nmArq;
	String proprietario;
	X509CertificateObject cert;

	for (int i = 0; i < arqs.length; i++) {

		//nome do arquivo
		nmArq = arqs[i].getName();

		//testa a extensao do arquivo
		if(nmArq.indexOf(".crt") > 0){

			try{
				//recupera o proprietario do certificado
				cert =
				Utils.getInstanceCertificadoX509(arqs[i].getAbsolut
				ePath());

				//compara apenas serial number porque o issuer já
				foi validado
				//anteriormente
				if(cert.getSerialNumber().equals(certTeste.getSerial
				Number())){
					return true;
				}


				}catch (Exception e) {
					//apenas nao adiciona se falhou
				}
			}
		}


		return false;
	}

}
```

Principal.Java

```
/**
 * Autor: Bruno
 *
 * Classe de tela principal do cliente.
 */
package cliente.ui;

import java.awt.GraphicsConfiguration;
import javax.swing.JFrame;

public class Principal extends JFrame {

	private javax.swing.JPanel jContentPane = null;
	private javax.swing.JLabel jLabel = null;
	private javax.swing.JTextField jTextField = null;
	private javax.swing.JButton jButton1 = null;
	private javax.swing.JButton jButton2 = null;
	private javax.swing.JButton jButton = null;
	private javax.swing.JButton jButton3 = null;
	private javax.swing.JButton jButton4 = null;


	public Principal() {
		super();
		initialize();
	}



	public Principal(GraphicsConfiguration arg0) {
		super(arg0);
		initialize();
	}


	public Principal(String arg0) {
		super(arg0);
		initialize();
	}



	public Principal(String arg0, GraphicsConfiguration arg1) {
		super(arg0, arg1);
		initialize();
	}

	public static void main(String[] args) {
	}


	private void initialize() {
		this.setContentPane(getJContentPane());
		this.setSize(413, 253);
		this.setTitle("Módulo Cliente");
	}
```

```
private javax.swing.JPanel getJContentPane() {
	if(jContentPane == null) {
		jContentPane = new javax.swing.JPanel();
		jContentPane.setLayout(null);
		jContentPane.add(getJLabel(), null);
		jContentPane.add(getJTextField(), null);
		jContentPane.add(getJButton1(), null);
		jContentPane.add(getJButton2(), null);
		jContentPane.add(getJButton(), null);
		jContentPane.add(getJButton3(), null);
		jContentPane.add(getJButton4(), null);
	}
	return jContentPane;
}


private javax.swing.JLabel getJLabel() {
	if(jLabel == null) {
		jLabel = new javax.swing.JLabel();
		jLabel.setBounds(4, 3, 379, 32);
		jLabel.setText("Projeto PKI - Módulo Cliente");
		jLabel.setFont(new java.awt.Font("Dialog",
		java.awt.Font.BOLD | java.awt.Font.ITALIC, 14));
		jLabel.setHorizontalAlignment(javax.swing.SwingConstants.
		LEFT);
		jLabel.setBackground(new java.awt.Color(3,22,191));
	}
	return jLabel;
}


private javax.swing.JTextField getJTextField() {
	if(jTextField == null) {
		jTextField = new javax.swing.JTextField();
		jTextField.setBounds(9, 49, 386, 24);
		jTextField.setText("Selecione uma Opção");
		jTextField.setBackground(new
		java.awt.Color(247,248,222));
		jTextField.setEnabled(false);
		jTextField.setFont(new java.awt.Font("Dialog",
		java.awt.Font.BOLD, 12));
		jTextField.setHorizontalAlignment(javax.swing.JTextField.
		CENTER);
	}
	return jTextField;
}


public javax.swing.JButton getJButton1() {
	if(jButton1 == null) {
		jButton1 = new javax.swing.JButton();
		jButton1.setBounds(8, 89, 160, 23);
		jButton1.setText("Importar Certificado");
	}
	return jButton1;
}


public javax.swing.JButton getJButton2() {
	if(jButton2 == null) {
		jButton2 = new javax.swing.JButton();
```

```
                jButton2.setBounds(8, 119, 160, 23);
                jButton2.setText("Exportar Certificado");
            }
            return jButton2;
        }



        public javax.swing.JButton getJButton() {
            if(jButton == null) {
                jButton = new javax.swing.JButton();
                jButton.setBounds(8, 149, 238, 23);
                jButton.setText("Assinar e Criptografar Mensagens");
            }
            return jButton;
        }



        public javax.swing.JButton getJButton3() {
            if(jButton3 == null) {
                jButton3 = new javax.swing.JButton();
                jButton3.setBounds(8, 179, 294, 23);
                jButton3.setText("Receber e Verificar Assinatura na
                Mensagem");
            }
            return jButton3;
        }



        public javax.swing.JButton getJButton4() {
            if(jButton4 == null) {
                jButton4 = new javax.swing.JButton();
                jButton4.setBounds(206, 91, 184, 21);
                jButton4.setText("Requisição de Certificado");
            }
            return jButton4;
        }
}
```

GeraCSR.Java

```
/**
 * Autor: Bruno
 *
 * Gera requisição de certificados no padrão PKCS #10
 *
 */
package cliente.ui;

import java.io.File;
import java.security.KeyPair;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import cliente.controle.ControleCliente;
import pki.Utils;
import util.GuiUtils;


public class GeraCSR extends JFrame {

	private javax.swing.JPanel jContentPane = null;
	private javax.swing.JLabel jLabel = null;
	private javax.swing.JTextField jTextField = null;
	private javax.swing.JLabel jLabel1 = null;
	private javax.swing.JLabel jLabel2 = null;
	private javax.swing.JTextField jTextField1 = null;
	private javax.swing.JButton jButton = null;
	private javax.swing.JTextField jTextField2 = null;


	public GeraCSR() {
		super();
		initialize();
	}


	private void initialize() {
		this.setSize(412, 249);
		this.setContentPane(getJContentPane());
	}


	private javax.swing.JPanel getJContentPane() {
		if (jContentPane == null) {
			jContentPane = new javax.swing.JPanel();
			jContentPane.setLayout(null);
			jContentPane.add(getJLabel(), null);
			jContentPane.add(getJTextField(), null);
			jContentPane.add(getJLabel1(), null);
			jContentPane.add(getJLabel2(), null);
			jContentPane.add(getJTextField1(), null);
			jContentPane.add(getJButton(), null);
			jContentPane.add(getJTextField2(), null);
		}
		return jContentPane;
	}
```

```
private javax.swing.JLabel getJLabel() {
      if(jLabel == null) {
            jLabel = new javax.swing.JLabel();
            jLabel.setBounds(8, 4, 384, 33);
            jLabel.setText("Projeto PKI - Módulo Cliente");
            jLabel.setFont(new java.awt.Font("Dialog",
            java.awt.Font.BOLD | java.awt.Font.ITALIC, 14));
            jLabel.addPropertyChangeListener(new
            java.beans.PropertyChangeListener() {
            public void propertyChange(java.beans.PropertyChangeEvent
            e) {
                  if ((e.getPropertyName().equals("text"))) {

                        System.out.println("propertyChange(text)");
                  }
            }
      });
   }
      return jLabel;
}


private javax.swing.JTextField getJTextField() {
      if(jTextField == null) {
            jTextField = new javax.swing.JTextField();
            jTextField.setBounds(72, 90, 296, 28);
      }
      return jTextField;
}


private javax.swing.JLabel getJLabel1() {
      if(jLabel1 == null) {
            jLabel1 = new javax.swing.JLabel();
            jLabel1.setBounds(13, 91, 57, 21);
            jLabel1.setText("Nome:");
      }
      return jLabel1;
}


private javax.swing.JLabel getJLabel2() {
      if(jLabel2 == null) {
            jLabel2 = new javax.swing.JLabel();
            jLabel2.setBounds(13, 125, 52, 22);
            jLabel2.setText("CPF:");
      }
      return jLabel2;
}


private javax.swing.JTextField getJTextField1() {
      if(jTextField1 == null) {
            jTextField1 = new javax.swing.JTextField();
            jTextField1.setBounds(72, 122, 296, 25);
      }
      return jTextField1;
}


private javax.swing.JButton getJButton() {
```

```
        if(jButton == null) {
              jButton = new javax.swing.JButton();
              jButton.setBounds(12, 172, 98, 25);
              jButton.setText("Gerar CSR");
              jButton.addActionListener(new
              java.awt.event.ActionListener() {
              public void actionPerformed(java.awt.event.ActionEvent e)
              {
                    gerarCSR();
              }
        });
    }
        return jButton;
}


/**
 * Gera requisição de certificado
 */
private void gerarCSR(){

        String nome = getJTextField().getText();
        String cpf = getJTextField1().getText();

        //valida os campos digitados pelo usuario
        if(nome.trim().length() == 0 || cpf.trim().length() == 0){
              JOptionPane.showMessageDialog(this, "Os campos não devem
              estar em branco.");
              return;
        }

        try{

              //nome que serah exibido no certificado
              String proprietario = ControleCliente.SUBJECT_CLIENTE +
              nome + ":" + cpf;

              //pergunta por um arquivo ao usuario
              File arq = GuiUtils.promptFileForSave(this, "Salvar
              Requisição de Certificado", null);
              if(arq == null){
                    //usuario abortou
                    return;
              }

              //gera par de chaves
              KeyPair kp = Utils.gerarKeyPairRSA();

              //gera CSR
              Utils.gerarCSR(arq.getAbsolutePath(), proprietario, kp,
              "SHA1withRSA");

              //grava apenas a chave privada, armazenamento temporario
              ateh a chegada do certificado
              Utils.gravarPrivKeyPkcs5(ControleCliente.ARQUIVO_CHAVE_TE
              MP_CLIENTE, ControleCliente.SENHA_CLIENTE.toCharArray(),
              kp.getPrivate());

              //ok
              JOptionPane.showMessageDialog(this, "Requisição de
              certificado gerado com sucesso!");
```

```
            }catch(Exception e){
                  e.printStackTrace();
                  JOptionPane.showMessageDialog(this, "Erro ao gerar a
                  requisição de certificado: "+e.getMessage());
            }
      }


      private javax.swing.JTextField getJTextField2() {
            if(jTextField2 == null) {
                  jTextField2 = new javax.swing.JTextField();
                  jTextField2.setBounds(9, 45, 382, 28);
                  jTextField2.setText("Dados do Cliente");
                  jTextField2.setBackground(new
                  java.awt.Color(247,248,222));
                  jTextField2.setEnabled(false);
                  jTextField2.setFont(new java.awt.Font("Dialog",
                  java.awt.Font.BOLD, 12));
                  jTextField2.setHorizontalAlignment(javax.swing.JTextField
                  .CENTER);
            }
            return jTextField2;
      }
}
```

Importacao.Java

```
/**
 * Autor: Bruno
 *
 * Classe para importação de certificados.
 * Verifica se o certificado importado faz parte de um
 * certificado de CA, um certificado pessoal ou de um
 * outro usuário.
 *
 */

package cliente.ui;

import java.awt.GraphicsConfiguration;
import javax.swing.JFrame;

public class Importacao extends JFrame {

	private javax.swing.JPanel jContentPane = null;
	private javax.swing.JLabel jLabel = null;
	private javax.swing.JTextField jTextField = null;
	private javax.swing.JComboBox jComboBox = null;
	private javax.swing.JButton jButton = null;


	public Importacao() {
		super();
		initialize();
	}



	public Importacao(GraphicsConfiguration arg0) {
		super(arg0);
		initialize();
	}



	public Importacao(String arg0) {
		super(arg0);
		initialize();
	}



	public Importacao(String arg0, GraphicsConfiguration arg1) {
		super(arg0, arg1);
		initialize();
	}

	public static void main(String[] args) {
	}


	private void initialize() {
		this.setSize(401, 250);
		this.setContentPane(getJContentPane());
	}
```

```
	private javax.swing.JPanel getJContentPane() {
		if (jContentPane == null) {
			jContentPane = new javax.swing.JPanel();
			jContentPane.setLayout(null);
			jContentPane.add(getJLabel(), null);
			jContentPane.add(getJTextField(), null);
			jContentPane.add(getJComboBox(), null);
			jContentPane.add(getJButton(), null);
		}
		return jContentPane;
	}


	private javax.swing.JLabel getJLabel() {
		if(jLabel == null) {
			jLabel = new javax.swing.JLabel();
			jLabel.setBounds(5, 3, 370, 31);
			jLabel.setText("Projeto PKI - Módulo Cliente");
			jLabel.setFont(new java.awt.Font("Dialog",
			java.awt.Font.BOLD | java.awt.Font.ITALIC, 14));
		}
		return jLabel;
	}

	private javax.swing.JTextField getJTextField() {
		if(jTextField == null) {
			jTextField = new javax.swing.JTextField();
			jTextField.setBounds(8, 48, 373, 26);
			jTextField.setText("Selecione o Certificado para
			importação");
			jTextField.setBackground(new
			java.awt.Color(248,249,226));
			jTextField.setEnabled(false);
			jTextField.setFont(new java.awt.Font("Dialog",
			java.awt.Font.BOLD, 12));
			jTextField.setHorizontalAlignment(javax.swing.JTextField.
			CENTER);
		}
		return jTextField;
	}
	private javax.swing.JComboBox getJComboBox() {
		if(jComboBox == null) {
			jComboBox = new javax.swing.JComboBox();
			jComboBox.setBounds(8, 96, 207, 21);

		}
		return jComboBox;
	}


	private javax.swing.JButton getJButton() {
		if(jButton == null) {
			jButton = new javax.swing.JButton();
			jButton.setBounds(278, 185, 97, 22);
			jButton.setText("Confirma");
		}
		return jButton;
	}
}
```

Exportacao.Java

```
/**
 * Autor: Bruno
 *
 * Classe para exportação de um certificado confiável
 *
 */
package cliente.ui;

import java.awt.GraphicsConfiguration;
import java.io.File;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import org.bouncycastle.jce.provider.X509CertificateObject;
import pki.Utils;
import util.FileUtils;
import util.GuiUtils;
import cliente.controle.ControleCliente;


public class Exportacao extends JFrame {

	private javax.swing.JPanel jContentPane = null;

	private javax.swing.JLabel jLabel = null;
	private javax.swing.JTextField jTextField = null;
	private javax.swing.JComboBox jComboBox = null;
	private javax.swing.JLabel jLabel1 = null;
	private javax.swing.JButton jButton = null;


	public Exportacao() {
		super();
		initialize();
	}



	public Exportacao(GraphicsConfiguration arg0) {
		super(arg0);
		initialize();
	}



	public Exportacao(String arg0) {
		super(arg0);
		initialize();
	}



	public Exportacao(String arg0, GraphicsConfiguration arg1) {
		super(arg0, arg1);
		initialize();
	}

	public static void main(String[] args) {
	}
```

```
private void initialize() {
	this.setSize(401, 250);
	this.setContentPane(getJContentPane());

	//carrega a lista de certificados
	listarCertificados();
}

/**
 * Carrega o combo com a lista de certificados.
 *
 */
private void listarCertificados(){

	//lista os certificados
	File dir = new File(ControleCliente.DIRETORIO_CERTIFICADOS);

	File arqs[] = dir.listFiles();
	String nmArq;
	String proprietario;
	X509CertificateObject cert;

	for (int i = 0; i < arqs.length; i++) {

		//nome do arquivo
		nmArq = arqs[i].getName();

		//testa a extensao do arquivo
		if(nmArq.indexOf(".crt") > 0){

			try{
				//recupera o proprietario do certificado
				cert =
				Utils.getInstanceCertificadoX509(arqs[i].getAb
				solutePath());
				proprietario = cert.getSubjectDN().getName();
				proprietario = Utils.extrairCN(proprietario);

				getJComboBox().addItem(nmArq + " - " +
				proprietario);

			}catch (Exception e) {
				//apenas nao adiciona se falhou
			}
		}
	}


}


private javax.swing.JPanel getJContentPane() {
	if (jContentPane == null) {
		jContentPane = new javax.swing.JPanel();
		jContentPane.setLayout(null);
		jContentPane.add(getJLabel(), null);
		jContentPane.add(getJTextField(), null);
		jContentPane.add(getJComboBox(), null);
		jContentPane.add(getJLabel1(), null);
```

```
            jContentPane.add(getJButton(), null);
      }
      return jContentPane;
}


private javax.swing.JLabel getJLabel() {
      if(jLabel == null) {
            jLabel = new javax.swing.JLabel();
            jLabel.setBounds(5, 3, 370, 31);
            jLabel.setText("Projeto PKI - Módulo Cliente");
            jLabel.setFont(new java.awt.Font("Dialog",
            java.awt.Font.BOLD | java.awt.Font.ITALIC, 14));
      }
      return jLabel;
}


private javax.swing.JTextField getJTextField() {
      if(jTextField == null) {
            jTextField = new javax.swing.JTextField();
            jTextField.setBounds(8, 48, 373, 26);
            jTextField.setText("Insira um Disquete para
            Exportação.");
            jTextField.setBackground(new
            java.awt.Color(248,249,226));
            jTextField.setEnabled(false);
            jTextField.setFont(new java.awt.Font("Dialog",
            java.awt.Font.BOLD, 12));
            jTextField.setHorizontalAlignment(javax.swing.JTextField.
            CENTER);
      }
      return jTextField;
}


private javax.swing.JComboBox getJComboBox() {
      if(jComboBox == null) {
            jComboBox = new javax.swing.JComboBox();
            jComboBox.setBounds(9, 121, 220, 23);
      }
      return jComboBox;
}


private javax.swing.JLabel getJLabel1() {
      if(jLabel1 == null) {
            jLabel1 = new javax.swing.JLabel();
            jLabel1.setBounds(9, 94, 155, 21);
            jLabel1.setText("Selecione o Certificado:");
      }
      return jLabel1;
}


private javax.swing.JButton getJButton() {
      if(jButton == null) {
            jButton = new javax.swing.JButton();
            jButton.setBounds(264, 187, 117, 24);
            jButton.setText("Exportar");
```

```
                jButton.addActionListener(new
                java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent e)
                {
                      exportarCertificado();
                }
          });
           }
          return jButton;
     }

     /**
      * Exporta o certificado selecionado.
      *
      */
     private void exportarCertificado(){

          String selecao = (String) getJComboBox().getSelectedItem();

          if(selecao != null){

                //recupera o nome do arquivo com a extensao
                int i = selecao.indexOf(".crt");
                String nmArq = selecao.substring(0, i + 4);
                String nmArqCer = ControleCliente.DIRETORIO_CERTIFICADOS
                + File.separator + nmArq;

                //copia certificado
                File nmArqCerDestino = GuiUtils.promptFileForSave(this,
                "Exportar certificado", null);

                if(nmArqCerDestino != null){

                      try{

                            FileUtils.copiarArquivo(new File(nmArqCer),
                            nmArqCerDestino);

                            //ok
                            JOptionPane.showMessageDialog(this,
                            "Certificado exportado com sucesso!");

                      }catch(Exception e){
                            e.printStackTrace();
                            JOptionPane.showMessageDialog(this, "Erro ao
                            exportar certificado: "+e.getMessage());
                      }
                }
          }

     }

}
```

EnviaMsg.java

```
/**
 * Autor: Bruno
 *
 * Classe para geração de mensagens assinadas e envelopadas
 * no padrão PKCS#7. As funcionalidades não se encontram
 * totalmente nesta classe.
 * Ela apenas utiliza os métodos de outras classes como a MsgUtils
 * e a Utils.
 *
 */
package cliente.ui;

import java.awt.GraphicsConfiguration;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.security.KeyPair;
import java.security.interfaces.RSAPrivateKey;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import org.bouncycastle.jce.provider.X509CertificateObject;
import pki.MsgUtils;
import pki.Utils;
import util.FileUtils;
import util.GuiUtils;
import cliente.controle.ControleCliente;


public class EnviaMsg extends JFrame {

	private javax.swing.JPanel jContentPane = null;

	private javax.swing.JLabel jLabel = null;
	private javax.swing.JTextField jTextField = null;
	private javax.swing.JTextArea jTextArea = null;
	private javax.swing.JButton jButton = null;
	private javax.swing.JLabel jLabel1 = null;
	private javax.swing.JComboBox jComboBox = null;


	public EnviaMsg() {
		super();
		initialize();
	}



	public EnviaMsg(GraphicsConfiguration arg0) {
		super(arg0);
		initialize();
	}



	public EnviaMsg(String arg0) {
		super(arg0);
		initialize();
```

```
}


public EnviaMsg(String arg0, GraphicsConfiguration arg1) {
	super(arg0, arg1);
	initialize();
}

public static void main(String[] args) {
}


private void initialize() {
	this.setSize(430, 322);
	this.setContentPane(getJContentPane());

	//listar certificados
	listarCertificados();
}

/**
 * Carrega o combo com a lista de certificados.
 */
private void listarCertificados(){

	//lista os certificados
	File dir = new File(ControleCliente.DIRETORIO_CERTIFICADOS);

	File arqs[] = dir.listFiles();
	String nmArq;
	String proprietario;
	X509CertificateObject cert;

	for (int i = 0; i < arqs.length; i++) {

		//nome do arquivo
		nmArq = arqs[i].getName();

		//testa a extensao do arquivo
		if(nmArq.indexOf(".crt") > 0){

			try{
				//recupera o proprietario do certificado
				cert =
				Utils.getInstanceCertificadoX509(arqs[i].getAb
				solutePath());
				proprietario = cert.getSubjectDN().getName();
				proprietario = Utils.extrairCN(proprietario);
				getJComboBox().addItem(nmArq + " - " +
				proprietario);

			}catch (Exception e) {
				//apenas nao adiciona se falhou
			}
		}
	}
}
```

```
private javax.swing.JPanel getJContentPane() {
	if (jContentPane == null) {
		jContentPane = new javax.swing.JPanel();
		jContentPane.setLayout(null);
		jContentPane.add(getJLabel(), null);
		jContentPane.add(getJTextField(), null);
		jContentPane.add(getJTextArea(), null);
		jContentPane.add(getJButton(), null);
		jContentPane.add(getJLabel1(), null);
		jContentPane.add(getJComboBox(), null);
	}
	return jContentPane;
}



private javax.swing.JLabel getJLabel() {
	if(jLabel == null) {
		jLabel = new javax.swing.JLabel();
		jLabel.setBounds(4, 3, 403, 31);
		jLabel.setText("Projeto PKI - Módulo Cliente");
		jLabel.setFont(new java.awt.Font("Dialog",
		java.awt.Font.BOLD | java.awt.Font.ITALIC, 14));
	}
	return jLabel;
}



private javax.swing.JTextField getJTextField() {
	if(jTextField == null) {
		jTextField = new javax.swing.JTextField();
		jTextField.setBounds(8, 44, 398, 25);
		jTextField.setBackground(new
		java.awt.Color(247,248,222));
		jTextField.setEnabled(false);
		jTextField.setFont(new java.awt.Font("Dialog",
		java.awt.Font.BOLD, 12));
		jTextField.setHorizontalAlignment(javax.swing.JTextField.
		CENTER);
		jTextField.setText("Texto a ser Cifrado e Assinado");
	}
	return jTextField;
}



private javax.swing.JTextArea getJTextArea() {
	if(jTextArea == null) {
		jTextArea = new javax.swing.JTextArea();
		jTextArea.setBounds(8, 74, 398, 139);
	}
	return jTextArea;
}



private javax.swing.JButton getJButton() {
	if(jButton == null) {
		jButton = new javax.swing.JButton();
```

```
            jButton.setBounds(8, 262, 147, 23);
            jButton.setText("Assinar e Cifrar");
            jButton.addActionListener(new
            java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e)
            {
                  enviarMensagem();
            }
      });
   }
      return jButton;
}

/**
 * Geração de mensagem assinada.
 *
 */
private void enviarMensagem(){

      String selecao = (String) getJComboBox().getSelectedItem();

      if(selecao != null){

            //recupera o nome do arquivo com a extensao
            int i = selecao.indexOf(".crt");
            String nmArq = selecao.substring(0, i + 4);
            String nmArqCer = ControleCliente.DIRETORIO_CERTIFICADOS
            + File.separator + nmArq;


            if(getJTextArea().getText().length() >
            0){//nmArqMsgOrigem != null){

                  try{

                        //le o arquivo a assinar
                        byte dados[] =
                        getJTextArea().getText().getBytes("ISO-8859-
                        1");

                        //variaveis necessarias
                        byte msgAssinada[] = null;
                        byte msgEnvelopada[] = null;

                        //certificado assinante
                        X509CertificateObject certAss =
                        Utils.getInstanceCertificadoX509(ControleClien
                        te.ARQUIVO_CERTIFICADO_CLIENTE);

                        //chave assinante
                        RSAPrivateKey privKey = (RSAPrivateKey)
                        Utils.lerPrivKeyPkcs12(ControleCliente.ARQUIVO
                        _CHAVE_CLIENTE,
                        ControleCliente.SENHA_CLIENTE.toCharArray(),
                        ControleCliente.ALIAS_CLIENTE);
                        KeyPair kp = new
                        KeyPair(certAss.getPublicKey(), privKey);

                        try{
                              //gera assinatura
```

```
            msgAssinada = MsgUtils.generatePkcs7SignedData(dados, kp,
            certAss);

            }catch(Exception e){
            throw new Exception("Erro ao gerar assinatura:
            "+e.getMessage());
            }

            try{

                  //certificado destinatario
                  X509CertificateObject certDest =
                  Utils.getInstanceCertificadoX509(nmArqCer);

                  //gera envelope
                  msgEnvelopada =
                  MsgUtils.generatePkcs7EnvelopedData(dados, certDest);

                  }catch(Exception e){
                        e.printStackTrace();
                        throw new Exception("Erro ao gerar envelope:
                        "+e.getMessage());
                  }

                  //arquivo de saida
                  File nmArqMsgSaida = GuiUtils.promptFileForSave(this,
                  "Arquivo de saída", null);


                  //grava a mensagem assinada e envelopada em sequencia
                  FileOutputStream fos = new
                  FileOutputStream(nmArqMsgSaida);
                  DataOutputStream dos = new DataOutputStream(fos);

                  //tamanho + mensagem assinada
                  dos.writeInt(msgAssinada.length);
                  dos.write(msgAssinada);

                  //tamanho + mensagem envelopada
                  dos.writeInt(msgEnvelopada.length);
                  dos.write(msgEnvelopada);

                  dos.flush();
                  fos.close();

                  //ok
                  JOptionPane.showMessageDialog(this, "Mensagem gerada com
                  sucesso!");

                  }catch(Exception e){
                        e.printStackTrace();
                        JOptionPane.showMessageDialog(this, "Erro ao gerar
                        a mensagem: "+e.getMessage());
                  }
                  }else{
                        JOptionPane.showMessageDialog(this, "Digite algo na
                        área de texto.");
                  }
            }

}
```

```
	private javax.swing.JLabel getJLabel1() {
		if(jLabel1 == null) {
			jLabel1 = new javax.swing.JLabel();
			jLabel1.setBounds(8, 231, 84, 23);
			jLabel1.setText("Enviar para:");
		}
		return jLabel1;
	}


	private javax.swing.JComboBox getJComboBox() {
		if(jComboBox == null) {
			jComboBox = new javax.swing.JComboBox();
			jComboBox.setBounds(97, 231, 217, 22);
		}
		return jComboBox;
	}
}
```

ReceberMsg.Java

```
/**
 * Autor: Bruno
 *
 * Classe que faz o procedimento de recepção de uma
 * mensagem assinada e envelopada.
 *
 */
package cliente.ui;

import java.awt.GraphicsConfiguration;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.security.KeyPair;
import java.security.cert.X509Certificate;
import java.security.interfaces.RSAPrivateKey;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import org.bouncycastle.jce.provider.X509CertificateObject;
import cliente.controle.ControleCliente;
import pki.MsgUtils;
import pki.Utils;
import util.GuiUtils;


public class ReceberMsg extends JFrame {

	private javax.swing.JPanel jContentPane = null;

	private javax.swing.JLabel jLabel = null;
	private javax.swing.JTextField jTextField = null;
	private javax.swing.JButton jButton = null;
	private javax.swing.JTextArea jTextArea = null;


	public ReceberMsg() {
		super();
		initialize();
	}


	public ReceberMsg(GraphicsConfiguration arg0) {
		super(arg0);
		initialize();
	}



	public ReceberMsg(String arg0) {
		super(arg0);
		initialize();
	}


	public ReceberMsg(String arg0, GraphicsConfiguration arg1) {
		super(arg0, arg1);
		initialize();
	}
```

```
public static void main(String[] args) {
}


private void initialize() {
	this.setSize(436, 215);
	this.setContentPane(getJContentPane());
}


private javax.swing.JPanel getJContentPane() {
	if (jContentPane == null) {
		jContentPane = new javax.swing.JPanel();
		jContentPane.setLayout(null);
		jContentPane.add(getJLabel(), null);
		jContentPane.add(getJTextField(), null);
		jContentPane.add(getJButton(), null);
		jContentPane.add(getJTextArea(), null);
	}
	return jContentPane;
}


private javax.swing.JLabel getJLabel() {
	if(jLabel == null) {
		jLabel = new javax.swing.JLabel();
		jLabel.setBounds(8, 7, 380, 31);
		jLabel.setText("Projeto PKI - Módulo Cliente");
		jLabel.setFont(new java.awt.Font("Dialog",
		java.awt.Font.BOLD | java.awt.Font.ITALIC, 14));
	}
	return jLabel;
}


private javax.swing.JTextField getJTextField() {
	if(jTextField == null) {
		jTextField = new javax.swing.JTextField();
		jTextField.setBounds(8, 44, 380, 25);
		jTextField.setBackground(new
		java.awt.Color(247,248,222));
		jTextField.setText("Abrir arquivo...");
		jTextField.setEnabled(false);
		jTextField.setFont(new java.awt.Font("Dialog",
		java.awt.Font.BOLD, 12));
	}
	return jTextField;
}


private javax.swing.JButton getJButton() {
	if(jButton == null) {
		jButton = new javax.swing.JButton();
		jButton.setBounds(10, 147, 206, 25);
		jButton.setText("Abrir e Conferir Assinatura");
		jButton.addActionListener(new
		java.awt.event.ActionListener() {
		public void actionPerformed(java.awt.event.ActionEvent e)
		{
			receberMensagem();
```

```
            }
        });
    }
        return jButton;
}

/**
 * Gera mensagem assinada.
 *
 */
private void receberMensagem(){

      //arquivo de origem
      File nmArqMsgOrigem = GuiUtils.promptFileForOpen(this, "Abrir
      arquivo", null);

      if(nmArqMsgOrigem != null){

            try{

                  //variaveis necessarias
                  byte msgAssinada[] = null;
                  byte msgEnvelopada[] = null;
                  byte dados[] = null;
                  String certRemetente;
                  String Remetente;

                  //arquivo de origem
                  FileInputStream fis = new
                  FileInputStream(nmArqMsgOrigem);
                  DataInputStream dis = new DataInputStream(fis);

                  msgAssinada = new byte[dis.readInt()];
                  dis.read(msgAssinada);

                  msgEnvelopada = new byte[dis.readInt()];
                  dis.read(msgEnvelopada);

                  fis.close();


                  //certificado local, supostamente o destinatario da
                  mensagem
                  X509CertificateObject certDest =
                  Utils.getInstanceCertificadoX509(ControleCliente.AR
                  QUIVO_CERTIFICADO_CLIENTE);
                  RSAPrivateKey privKey = (RSAPrivateKey)
                  Utils.lerPrivKeyPkcs12(ControleCliente.ARQUIVO_CHAV
                  E_CLIENTE,
                  ControleCliente.SENHA_CLIENTE.toCharArray(),
                  ControleCliente.ALIAS_CLIENTE);


            try{

                  //abre envelope
                  dados =
                  MsgUtils.openPkcs7EnvelopedData(msgEnvelopada,
                  privKey);

                  //exibe a mensagem recuperada
```

```
                    //getJTextArea().setText(new String(dados, "ISO-
                    8859-1") );


                    }catch(Exception e){
                          e.printStackTrace();
                          throw new Exception("Erro ao abrir envelope:
                          "+e.getMessage());
                    }

try{
      //descobre o certificado de origem da mensagem para validar a
      assinatura
      X509Certificate certAssinante =
      MsgUtils.getCertificateFromPkcs7SignedData(msgAssinada);
      certRemetente = certAssinante.getSubjectDN().toString();
      Remetente = Utils.extrairCN(certRemetente);
      //Apresenta msg recebida e o remetente da mesma.
      getJTextArea().setText("Remetente: " + Remetente + "\nMensagem:
      " + new String(dados, "ISO-8859-1") );

      //valida AC
      if(!ControleCliente.validarRoot(this, certAssinante)){
            //se falhou cancela
            return;
      }

      //verifica se o certificado foi revogado
      if(ControleCliente.isCertificadoRevogado(this, certAssinante)){
            //se falhou cancela
            JOptionPane.showMessageDialog(this, "O certificado do
            assinante foi revogado, operação cancelada!");
            getJTextArea().setText("");
            return;
      }


      //verifica assinatura
      if(MsgUtils.verifyPkcs7SignedData(msgAssinada, dados,
      certAssinante)){

            //ok
            JOptionPane.showMessageDialog(this, "Assinatura
            verificada com sucesso!");
      }else{
            JOptionPane.showMessageDialog(this, "Assinatura NÃO
            confere!");
      }

}catch(Exception e){
      throw new Exception("Erro ao verificar assinatura:
      "+e.getMessage());
}
 }catch(Exception e){
      e.printStackTrace();
      JOptionPane.showMessageDialog(this, "Erro ao abrir a mensagem:
      "+e.getMessage());
  }
}

}
```

```
	private javax.swing.JTextArea getJTextArea() {
		if(jTextArea == null) {
			jTextArea = new javax.swing.JTextArea();
			jTextArea.setBounds(11, 73, 377, 58);
			jTextArea.setWrapStyleWord(true);
			jTextArea.setEditable(false);
		}
		return jTextArea;
	}
}
```

ANEXO C – Métodos para gerenciamento da ICP

Base64Decoder.Java

```
package pki;

// Base64Decoder.java
// $Id: Base64Decoder.java,v 1.5 2000/06/08 16:36:05 ylafon Exp $
// (c) COPYRIGHT MIT and INRIA, 1996.
// Please first read the full copyright statement in file COPYRIGHT.html

import java.io.*;

/**
 * Decode a BASE64 encoded input stream to some output stream.
 * This class implements BASE64 decoding, as specified in the
 * <a href="http://ds.internic.net/rfc/rfc1521.txt">MIME specification</a>.
 * @see org.w3c.tools.codec.Base64Encoder
 *
 * Nota: Classe oferecida pela organizacao
 * Bouncy Castle para decodificacao de dados
 * em Base64.
 *
 *
 */

public class Base64Decoder {
      private static final int BUFFER_SIZE = 1024 ;

      InputStream in       = null ;
      OutputStream out     = null ;
      boolean      stringp = false ;

      /**
       * Create a decoder to decode a stream.
       * @param in The input stream (to be decoded).
       * @param out The output stream, to write decoded data to.
       */

      public Base64Decoder (InputStream in, OutputStream out) {
      this.in = in ;
      this.out = out ;
      this.stringp = false ;
      }
      /**
       * Create a decoder to decode a String.
       * @param input The string to be decoded.
       */

      public Base64Decoder (String input) {
      byte bytes[] ;
      try {
          bytes = input.getBytes ("ISO-8859-1");
      } catch (UnsupportedEncodingException ex) {
          throw new RuntimeException(this.getClass().getName() +
                                     "[Constructor] Unable to convert" +
                                     "properly char to bytes");
      }
      this.stringp = true ;
```

```
this.in      = new ByteArrayInputStream(bytes) ;
this.out     = new ByteArrayOutputStream () ;
}
private final int check (int ch) {
if ((ch >= 'A') && (ch <= 'Z')) {
    return ch - 'A' ;
} else if ((ch >= 'a') && (ch <= 'z')) {
    return ch - 'a' + 26 ;
} else if ((ch >= '0') && (ch <= '9')) {
    return ch - '0' + 52 ;
} else {
    switch (ch) {
      case '=':
        return 65 ;
      case '+':
        return 62 ;
      case '/':
        return 63 ;
      default:
        return -1 ;
    }
}
}
private final int get1 (byte buf[], int off) {
return ((buf[off] & 0x3f) << 2) | ((buf[off+1] & 0x30) >>> 4) ;
}
private final int get2 (byte buf[], int off) {
return ((buf[off+1] & 0x0f) << 4) | ((buf[off+2] &0x3c) >>> 2) ;
}
private final int get3 (byte buf[], int off) {
return ((buf[off+2] & 0x03) << 6) | (buf[off+3] & 0x3f) ;
}
/**
 * Test the decoder.
 * Run it with one argument: the string to be decoded, it will print
 * out
 * the decoded value.
 */

public static void main (String args[]) {
if ( args.length == 1 ) {
    try {
      Base64Decoder b = new Base64Decoder (args[0]) ;
      System.out.println ("["+b.processString()+"]") ;
    } catch (Base64FormatException e) {
      System.out.println ("Invalid Base64 format !") ;
      System.exit (1) ;
    }
} else if ((args.length == 2) && (args[0].equals("-f"))) {
    try {
      FileInputStream in = new FileInputStream(args[1]) ;
      Base64Decoder b = new Base64Decoder (in, System.out);
      b.process();
    } catch (Exception ex) {
      System.out.println("error: " + ex.getMessage());
      System.exit(1) ;
    }
} else {
    System.out.println("Base64Decoder [strong] [-f file]");
}
System.exit (0) ;
```

```
}
private void printHex (byte buf[], int off, int len) {
while (off < len) {
    printHex (buf[off++]) ;
    System.out.print (" ") ;
}
System.out.println ("") ;
}
private void printHex (int x) {
int h = (x&0xf0) >> 4 ;
int l = (x&0x0f) ;
System.out.print
    ((new Character((char)((h>9) ? 'A'+h-10 : '0'+h))).toString()
     +(new Character((char)((l>9) ? 'A'+l-10 : '0'+l))).toString());
}
private void printHex (String s) {
byte bytes[] ;
try {
    bytes = s.getBytes ("ISO-8859-1");
} catch (UnsupportedEncodingException ex) {
    throw new RuntimeException(this.getClass().getName() +
                          "[printHex] Unable to convert" +
                          "properly char to bytes");
}
printHex (bytes, 0, bytes.length) ;
}
/**
 * Do the actual decoding.
 * Process the input stream by decoding it and emiting the resulting
bytes
 * into the output stream.
 * @exception IOException If the input or output stream accesses
failed.
 * @exception Base64FormatException If the input stream is not
compliant
 *    with the BASE64 specification.
 */

public void process ()
throws IOException, Base64FormatException
{
byte buffer[] = new byte[BUFFER_SIZE] ;
byte chunk[]  = new byte[4] ;
int  got      = -1 ;
int  ready    = 0 ;

fill:
while ((got = in.read(buffer)) > 0) {
    int skiped = 0 ;
    while ( skiped < got ) {
      // Check for un-understood characters:
      while ( ready < 4 ) {
          if ( skiped >= got )
            continue fill ;
          int ch = check (buffer[skiped++]) ;
          if ( ch >= 0 )
            chunk[ready++] = (byte) ch ;
      }
      if ( chunk[2] == 65 ) {
          out.write(get1(chunk, 0));
          return ;
```

```
            } else if ( chunk[3] == 65 ) {
                out.write(get1(chunk, 0)) ;
                out.write(get2(chunk, 0)) ;
                return ;
            } else {
                out.write(get1(chunk, 0)) ;
                out.write(get2(chunk, 0)) ;
                out.write(get3(chunk, 0)) ;
            }
            ready = 0 ;
          }
      }
      if ( ready != 0 )
          throw new Base64FormatException ("Invalid length.") ;
      out.flush() ;
      }
      /**
       * Do the decoding, and return a String.
       * This methods should be called when the decoder is used in
       * <em>String</em> mode. It decodes the input string to an output
string
       * that is returned.
       * @exception RuntimeException If the object wasn't constructed to
       *    decode a String.
       * @exception Base64FormatException If the input string is not
compliant
       *     with the BASE64 specification.
       */

      public String processString ()
      throws Base64FormatException
      {
      if ( ! stringp )
          throw new RuntimeException (this.getClass().getName()
                                + "[processString]"
                                + "invalid call (not a String)");
      try {
          process() ;
      } catch (IOException e) {
      }
      String s;
      try {
          s = ((ByteArrayOutputStream) out).toString("ISO-8859-1") ;
      } catch (UnsupportedEncodingException ex) {
          throw new RuntimeException(this.getClass().getName() +
                                "[processString] Unable to convert" +
                                "properly char to bytes");
      }
      return s;
      }
}
```

Base64Encoder.Java

```
package pki;

// Base64Encoder.java
// $Id: Base64Encoder.java,v 1.7 2000/06/08 16:35:01 ylafon Exp $
// (c) COPYRIGHT MIT and INRIA, 1996.
// Please first read the full copyright statement in file COPYRIGHT.html

import java.io.*;

/**
 * BASE64 encoder implementation.
 * This object takes as parameter an input stream and an output stream. It
 * encodes the input stream, using the BASE64 encoding rules, as defined
 * in <a href="http://ds.internic.net/rfc/rfc1521.txt">MIME
specification</a>
 * and emit the resulting data to the output stream.
 * @see org.w3c.tools.codec.Base64Decoder
 *
 *
 * Nota: Classe oferecida pela organizacao
 * Bouncy Castle para codificacao de dados
 * em Base64.
 *
 */

public class Base64Encoder {
      private static final int BUFFER_SIZE = 1024 ;
      private static byte encoding[] =
      {
      (byte) 'A', (byte) 'B', (byte) 'C', (byte) 'D',
      (byte) 'E', (byte) 'F', (byte) 'G', (byte) 'H',       // 0-7
      (byte) 'I', (byte) 'J', (byte) 'K', (byte) 'L',
      (byte) 'M', (byte) 'N', (byte) 'O', (byte) 'P',       // 8-15
      (byte) 'Q', (byte) 'R', (byte) 'S', (byte) 'T',
      (byte) 'U', (byte) 'V', (byte) 'W', (byte) 'X',       // 16-23
      (byte) 'Y', (byte) 'Z', (byte) 'a', (byte) 'b',
      (byte) 'c', (byte) 'd', (byte) 'e', (byte) 'f',       // 24-31
      (byte) 'g', (byte) 'h', (byte) 'i', (byte) 'j',
      (byte) 'k', (byte) 'l', (byte) 'm', (byte) 'n',       // 32-39
      (byte) 'o', (byte) 'p', (byte) 'q', (byte) 'r',
      (byte) 's', (byte) 't', (byte) 'u', (byte) 'v',       // 40-47
      (byte) 'w', (byte) 'x', (byte) 'y', (byte) 'z',
      (byte) '0', (byte) '1', (byte) '2', (byte) '3',       // 48-55
      (byte) '4', (byte) '5', (byte) '6', (byte) '7',
      (byte) '8', (byte) '9', (byte) '+', (byte) '/',       // 56-63
      (byte) '='                                    // 64
      };

      InputStream  in  = null ;
      OutputStream out = null ;
      boolean      stringp = false ;

      /**
       * Create a new Base64 encoder, encoding input to output.
       * @param in The input stream to be encoded.
       * @param out The output stream, to write encoded data to.
       */
```

```
public Base64Encoder (InputStream in, OutputStream out) {
this.in      = in ;
this.out     = out ;
this.stringp = false ;
}
/**
 * Create a new Base64 encoder, to encode the given string.
 * @param input The String to be encoded.
 */

public Base64Encoder (String input) {
byte bytes[] ;
try {
    bytes = input.getBytes ("ISO-8859-1");
} catch (UnsupportedEncodingException ex) {
    throw new RuntimeException(this.getClass().getName() +
                          "[Constructor] Unable to convert" +
                          "properly char to bytes");
}
this.stringp = true ;
this.in      = new ByteArrayInputStream(bytes) ;
this.out     = new ByteArrayOutputStream () ;
}
private final int get1(byte buf[], int off) {
return (buf[off] & 0xfc) >> 2 ;
}
private final int get2(byte buf[], int off) {
return ((buf[off]&0x3) << 4) | ((buf[off+1]&0xf0) >>> 4) ;
}
private final int get3(byte buf[], int off) {
return ((buf[off+1] & 0x0f) << 2) | ((buf[off+2] & 0xc0) >>> 6) ;
}
private static final int get4(byte buf[], int off) {
return buf[off+2] & 0x3f ;
}
/**
 * Testing the encoder.
 * Run with one argument, prints the encoded version of it.
 */

public static void main (String args[]) {
if ( args.length != 1 ) {
    System.out.println ("Base64Encoder <string>") ;
    System.exit (0) ;
}
Base64Encoder b = new Base64Encoder (args[0]) ;
System.out.println ("["+b.processString()+"]") ;
// joe:eoj -> am9lOmVvag==
// 12345678:87654321 -> MTIzNDU2Nzg6ODc2NTQzMjE=
}
/**
 * Process the data: encode the input stream to the output stream.
 * This method runs through the input stream, encoding it to the
output
 * stream.
 * @exception IOException If we weren't able to access the input
stream or
 *    the output stream.
 */
```

```
public void process ()
      throws IOException
      {
      byte buffer[] = new byte[BUFFER_SIZE] ;
      int  got      = -1 ;
      int  off      = 0 ;
      int  count    = 0 ;
      while ((got = in.read(buffer, off, BUFFER_SIZE-off)) > 0) {
          if ( got >= 3 ) {
            got += off;
            off  = 0;
            while (off + 3 <= got) {
                int c1 = get1(buffer,off) ;
                int c2 = get2(buffer,off) ;
                int c3 = get3(buffer,off) ;
                int c4 = get4(buffer,off) ;
                switch (count) {
                  case 73:
                    out.write(encoding[c1]);
                    out.write(encoding[c2]);
                    out.write(encoding[c3]);
                    out.write ('\n') ;
                    out.write(encoding[c4]) ;
                    count = 1 ;
                    break ;
                  case 74:
                    out.write(encoding[c1]);
                    out.write(encoding[c2]);
                    out.write ('\n') ;
                    out.write(encoding[c3]);
                    out.write(encoding[c4]) ;
                    count = 2 ;
                    break ;
                  case 75:
                    out.write(encoding[c1]);
                    out.write ('\n') ;
                    out.write(encoding[c2]);
                    out.write(encoding[c3]);
                    out.write(encoding[c4]) ;
                    count = 3 ;
                    break ;
                  case 76:
                    out.write('\n') ;
                    out.write(encoding[c1]);
                    out.write(encoding[c2]);
                    out.write(encoding[c3]);
                    out.write(encoding[c4]) ;
                    count = 4 ;
                    break ;
                  default:
                    out.write(encoding[c1]);
                    out.write(encoding[c2]);
                    out.write(encoding[c3]);
                    out.write(encoding[c4]) ;
                    count += 4 ;
                    break ;
                }
                off += 3 ;
            }
            // Copy remaining bytes to beginning of buffer:
            for ( int i = 0 ; i < 3 ;i++)
```

```
                buffer[i] = (i < got-off) ? buffer[off+i] : ((byte) 0) ;
              off = got-off ;
            } else {
              // Total read amount is less then 3 bytes:
              off += got;
            }
        }
        // Manage the last bytes, from 0 to off:
        switch (off) {
          case 1:
              out.write(encoding[get1(buffer, 0)]) ;
              out.write(encoding[get2(buffer, 0)]) ;
              out.write('=') ;
              out.write('=') ;
              break ;
          case 2:
              out.write(encoding[get1(buffer, 0)]);
              out.write(encoding[get2(buffer, 0)]);
              out.write(encoding[get3(buffer, 0)]);
              out.write('=');
        }
        return ;
        }
        /**
         * Encode the content of this encoder, as a string.
         * This methods encode the String content, that was provided at
        creation
         * time, following the BASE64 rules, as specified in the rfc1521.
         * @return A String, reprenting the encoded content of the input
        String.
         */

        public String processString () {
        if ( ! stringp )
            throw new RuntimeException (this.getClass().getName()
                                + "[processString]"
                                + "invalid call (not a String)");
        try {
            process() ;
        } catch (IOException e) {
        }
        return ((ByteArrayOutputStream) out).toString() ;
        }
}
```

Base64FormatException.Java

```
package pki;

// Base64FormatException.java
// $Id: Base64FormatException.java,v 1.3 1998/01/22 12:42:16 bmahe Exp $
// (c) COPYRIGHT MIT and INRIA, 1996.
// Please first read the full copyright statement in file COPYRIGHT.html

// Base64FormatException.java
// $Id: Base64FormatException.java,v 1.3 1998/01/22 12:42:16 bmahe Exp $
// (c) COPYRIGHT MIT and INRIA, 1996.
// Please first read the full copyright statement in file COPYRIGHT.html

/**
 * Exception for invalid BASE64 streams.
 *
 * Nota: Classe oferecida pela organizacao
 * Bouncy Castle para tratar exceptions.
 *
 *
 */

public class Base64FormatException extends Exception {

  /**
   * Create that kind of exception
   * @param msg The associated error message
   */

  public Base64FormatException(String msg) {
      super(msg) ;
  }
}
```

UtilRSAPrivateCrtKey.Java

```
/**
 * A provider representation for a RSA private key, with CRT factors
included.
 *
 * Nota: Classe oferecida pela RSA para gerenciamento
 * de chaves privadas utilizadas na criptografia.
 * Aceita passagem de parametros.
 *
 *
 */
package pki;

import java.io.*;
import java.math.BigInteger;
import java.security.spec.RSAPrivateCrtKeySpec;
import java.security.interfaces.RSAPrivateCrtKey;
import org.bouncycastle.asn1.*;
import org.bouncycastle.asn1.x509.AlgorithmIdentifier;
import org.bouncycastle.asn1.pkcs.PrivateKeyInfo;
import org.bouncycastle.asn1.pkcs.PKCSObjectIdentifiers;
import org.bouncycastle.asn1.pkcs.RSAPrivateKeyStructure;
import org.bouncycastle.jce.provider.JCERSAPrivateKey;


public class UtilRSAPrivateCrtKey
      extends JCERSAPrivateKey
      implements RSAPrivateCrtKey
{
      private BigInteger  publicExponent;
      private BigInteger  primeP;
      private BigInteger  primeQ;
      private BigInteger  primeExponentP;
      private BigInteger  primeExponentQ;
      private BigInteger  crtCoefficient;

/**
 * construct an RSA key from a private key info object.
 */
public UtilRSAPrivateCrtKey(PrivateKeyInfo info) {
      this(new RSAPrivateKeyStructure((DERConstructedSequence)
info.getPrivateKey()));
}
/**
 * construct a private key from another RSAPrivateCrtKey.
 *
 * @param key the object implementing the RSAPrivateCrtKey interface.
 */
public UtilRSAPrivateCrtKey(RSAPrivateCrtKey key) {
      this.modulus = key.getModulus();
      this.publicExponent = key.getPublicExponent();
      this.privateExponent = key.getPrivateExponent();
      this.primeP = key.getPrimeP();
      this.primeQ = key.getPrimeQ();
      this.primeExponentP = key.getPrimeExponentP();
      this.primeExponentQ = key.getPrimeExponentQ();
      this.crtCoefficient = key.getCrtCoefficient();
}
/**
```

```
     * construct a private key from an RSAPrivateCrtKeySpec
     *
     * @param spec the spec to be used in construction.
     */
    public UtilRSAPrivateCrtKey(RSAPrivateCrtKeySpec spec) {
          this.modulus = spec.getModulus();
          this.publicExponent = spec.getPublicExponent();
          this.privateExponent = spec.getPrivateExponent();
          this.primeP = spec.getPrimeP();
          this.primeQ = spec.getPrimeQ();
          this.primeExponentP = spec.getPrimeExponentP();
          this.primeExponentQ = spec.getPrimeExponentQ();
          this.crtCoefficient = spec.getCrtCoefficient();
    }
    /**
     * construct an RSA key from a ASN.1 RSA private key object.
     */
    public UtilRSAPrivateCrtKey(RSAPrivateKeyStructure key) {
          this.modulus = key.getModulus();
          this.publicExponent = key.getPublicExponent();
          this.privateExponent = key.getPrivateExponent();
          this.primeP = key.getPrime1();
          this.primeQ = key.getPrime2();
          this.primeExponentP = key.getExponent1();
          this.primeExponentQ = key.getExponent2();
          this.crtCoefficient = key.getCoefficient();
    }
          public boolean equals(Object o)
          {
                if ( !(o instanceof RSAPrivateCrtKey) )
                {
                      return false;
                }

                if ( o == this )
                {
                      return true;
                }

                RSAPrivateCrtKey key = (RSAPrivateCrtKey)o;

                return this.getModulus().equals(key.getModulus())
                 && this.getPublicExponent().equals(key.getPublicExponent())
                 && this.getPrivateExponent().equals(key.getPrivateExponent())
                 && this.getPrimeP().equals(key.getPrimeP())
                 && this.getPrimeQ().equals(key.getPrimeQ())
                 && this.getPrimeExponentP().equals(key.getPrimeExponentP())
                 && this.getPrimeExponentQ().equals(key.getPrimeExponentQ())
                 && this.getCrtCoefficient().equals(key.getCrtCoefficient());
          }
          /**
           * return the CRT coefficient.
           *
           * @return the CRT coefficient.
           */
          public BigInteger getCrtCoefficient()
          {
                return crtCoefficient;
          }
          /**
           * Return a PKCS8 representation of the key. The sequence returned
```

```
     * represents a full PrivateKeyInfo object.
     *
     * @return a PKCS8 representation of the key.
     */
    public byte[] getEncoded()
    {
          ByteArrayOutputStream   bOut = new ByteArrayOutputStream();
          DEROutputStream         dOut = new DEROutputStream(bOut);
          PrivateKeyInfo          info = new PrivateKeyInfo(new
          AlgorithmIdentifier(PKCSObjectIdentifiers.rsaEncryption, null),
          new RSAPrivateKeyStructure(getModulus(), getPublicExponent(),
          getPrivateExponent(), getPrimeP(), getPrimeQ(),
          getPrimeExponentP(), getPrimeExponentQ(),
          getCrtCoefficient()).getDERObject());

          try
          {
                dOut.writeObject(info);
                dOut.close();
          }
          catch (IOException e)
          {
                throw new RuntimeException("Error encoding RSA public
                key");
          }

          return bOut.toByteArray();
    }
    /**
     * return the encoding format we produce in getEncoded().
     *
     * @return the encoding format we produce in getEncoded().
     */
    public String getFormat()
    {
          return "PKCS#8";
    }
    /**
     * return the prime exponent for P.
     *
     * @return the prime exponent for P.
     */
    public BigInteger getPrimeExponentP()
    {
          return primeExponentP;
    }
    /**
     * return the prime exponent for Q.
     *
     * @return the prime exponent for Q.
     */
    public BigInteger getPrimeExponentQ()
    {
          return primeExponentQ;
    }
    /**
     * return the prime P.
     *
     * @return the prime P.
     */
    public BigInteger getPrimeP()
```

```
        {
            return primeP;
        }
        /**
         * return the prime Q.
         *
         * @return the prime Q.
         */
        public BigInteger getPrimeQ()
        {
            return primeQ;
        }
        /**
         * return the public exponent.
         *
         * @return the public exponent.
         */
        public BigInteger getPublicExponent()
        {
            return publicExponent;
        }
        public String toString()
        {
            StringBuffer    buf = new StringBuffer();
            String          nl = System.getProperty("line.separator");

            buf.append("RSA Private CRT Key" + nl);
            buf.append("            modulus: " +
            this.getModulus().toString(16) + nl);
            buf.append("    public exponent: " +
            this.getPublicExponent().toString(16) + nl);
            buf.append("   private exponent: " +
            this.getPrivateExponent().toString(16) + nl);
            buf.append("             primeP: " +
            this.getPrimeP().toString(16) + nl);
            buf.append("             primeQ: " +
            this.getPrimeQ().toString(16) + nl);
            buf.append("     primeExponentP: " +
            this.getPrimeExponentP().toString(16) + nl);
            buf.append("     primeExponentQ: " +
            this.getPrimeExponentQ().toString(16) + nl);
            buf.append("     crtCoefficient: " +
            this.getCrtCoefficient().toString(16) + nl);

            return buf.toString();
        }
}
```

UtilRSAPrivateKey.Java

```
**
 * Chave privada sem CRT parameters.
 *
 * Nota: Classe oferecida pela RSA para gerenciamento
 * de chaves privadas utilizadas na criptografia.
 * Nao precisa de parametros para gerenciamento.
 *
 */
package pki;

import java.io.*;
import java.math.BigInteger;
import java.security.interfaces.RSAPrivateKey;
import org.bouncycastle.asn1.*;
import org.bouncycastle.asn1.x509.AlgorithmIdentifier;
import org.bouncycastle.asn1.pkcs.PrivateKeyInfo;
import org.bouncycastle.asn1.pkcs.PKCSObjectIdentifiers;
import org.bouncycastle.asn1.pkcs.RSAPrivateKeyStructure;
import org.bouncycastle.crypto.params.RSAKeyParameters;
import java.security.spec.RSAPrivateKeySpec;


public class UtilRSAPrivateKey
      implements RSAPrivateKey
{
      private BigInteger  publicExponent;
      private BigInteger modulo;
      private BigInteger privExp;

public UtilRSAPrivateKey(BigInteger modulus, BigInteger privExponent) {
      this.modulo = modulus;
      this.publicExponent = null;
      this.privExp=privExponent;

}
public UtilRSAPrivateKey(BigInteger modulus, BigInteger publicExponent,
BigInteger privExponent) {
      this.modulo = modulus;
      this.publicExponent = publicExponent;
      this.privExp=privExponent;

}
public UtilRSAPrivateKey(RSAKeyParameters key) {
      this.modulo = key.getModulus();
      this.privExp = key.getExponent();
}
public UtilRSAPrivateKey(RSAPrivateKey key) {
      this.modulo = key.getModulus();
      this.privExp = key.getPrivateExponent();
}
public UtilRSAPrivateKey(RSAPrivateKeySpec spec) {
      this.modulo = spec.getModulus();
      this.privExp = spec.getPrivateExponent();
}
            public String getAlgorithm(){
                  return "RSA";
            }
public byte[] getEncoded() {
```

```
            BigInteger zero=new BigInteger("00", 16);

            ByteArrayOutputStream   bOut = new ByteArrayOutputStream();
            DEROutputStream         dOut = new DEROutputStream(bOut);
            PrivateKeyInfo          info = new PrivateKeyInfo(new
            AlgorithmIdentifier(PKCSObjectIdentifiers.rsaEncryption, null),
            new RSAPrivateKeyStructure(getModulus(), getPublicExponent(),
            getPrivateExponent(), zero, zero, zero, zero,
            zero).getDERObject());

            try
            {
                  dOut.writeObject(info);
                  dOut.close();
            }
            catch (IOException e)
            {
                  throw new RuntimeException("Error encoding RSA public
                  key");
            }

            return bOut.toByteArray();
}
public String getFormat() {
      return "PKCS#8";
//    return "NULL";
}
            public BigInteger getModulus(){
                  return modulo;
            }
            public BigInteger getPrivateExponent(){
                  return privExp;
            }
      /**
       * return the public exponent.
       *
       * @return the public exponent.
       */
      public BigInteger getPublicExponent()
      {
            return publicExponent;
      }
            /**
       * Generate a String representation of this key.
       *
       * @return        The key as a string.
       */
      public String toString()
      {
            return modulo.toString(16) + "." + privExp.toString(16);
      }
}
```

UtilRSAPublicKey.Java

```
/**
 * Insert the type's description here.
 * Creation date: (7/6/2002 08:24:31)
 *
 * Nota: Classe oferecida pela RSA para gerenciamento
 * de chaves publicas utilizadas na criptografia.
 *
 */
package pki;

import java.io.*;
import java.math.BigInteger;
import java.security.interfaces.RSAPublicKey;
import org.bouncycastle.asn1.DEROutputStream;
import org.bouncycastle.asn1.pkcs.PKCSObjectIdentifiers;
import org.bouncycastle.asn1.x509.AlgorithmIdentifier;
import org.bouncycastle.asn1.x509.SubjectPublicKeyInfo;
import org.bouncycastle.asn1.x509.RSAPublicKeyStructure;


public class UtilRSAPublicKey implements RSAPublicKey {
            BigInteger modulus;
            BigInteger publicExponent;


/**
 * Insert the method's description here.
 * Creation date: (7/6/2002 08:24:59)
 */
public UtilRSAPublicKey() {


      }
/**
 * Insert the method's description here.
 * Creation date: (7/6/2002 08:24:59)
 */
public UtilRSAPublicKey(        BigInteger modulus,      BigInteger
publicExponent) {
      this.modulus=modulus;
      this.publicExponent=publicExponent;

      }
            public boolean equals(Object o)
      {
            if ( !(o instanceof RSAPublicKey) )
            {
                  return false;
            }

            if ( o == this )
            {
                  return true;
            }

            RSAPublicKey key = (RSAPublicKey)o;

            return getModulus().equals(key.getModulus())
```

```
            && getPublicExponent().equals(key.getPublicExponent());
      }
      /**
       * Returns the standard algorithm name for this key. For
       * example, "DSA" would indicate that this key is a DSA key.
       * See Appendix A in the <a href=
       * "../../../guide/security/CryptoSpec.html#AppA">
       * Java Cryptography Architecture API Specification &amp; Reference
</a>
       * for information about standard algorithm names.
       *
       * @return the name of the algorithm associated with this key.
       */
public java.lang.String getAlgorithm() {
            return "RSA";
}
      /**
       * Returns the key in its primary encoding format, or null
       * if this key does not support encoding.
       *
       * @return the encoded key, or null if the key does not support
       * encoding.
       */
public byte[] getEncoded() {
            ByteArrayOutputStream   bOut = new ByteArrayOutputStream();
            DEROutputStream         dOut = new DEROutputStream(bOut);
            SubjectPublicKeyInfo    info = new SubjectPublicKeyInfo(new
            AlgorithmIdentifier(PKCSObjectIdentifiers.rsaEncryption, null),
            new RSAPublicKeyStructure(getModulus(),
            getPublicExponent()).getDERObject());

            try
            {
                  dOut.writeObject(info);
                  dOut.close();
            }
            catch (IOException e)
            {
                  throw new RuntimeException("Error encoding RSA public
                  key");
            }

            return bOut.toByteArray();
}
      /**
       * Returns the name of the primary encoding format of this key,
       * or null if this key does not support encoding.
       * The primary encoding format is
       * named in terms of the appropriate ASN.1 data format, if an
       * ASN.1 specification for this key exists.
       * For example, the name of the ASN.1 data format for public
       * keys is <I>SubjectPublicKeyInfo</I>, as
       * defined by the X.509 standard; in this case, the returned format
      is
       * <code>"X.509"</code>. Similarly,
       * the name of the ASN.1 data format for private keys is
       * <I>PrivateKeyInfo</I>,
       * as defined by the PKCS #8 standard; in this case, the returned
      format is
       * <code>"PKCS#8"</code>.
       *
```

```
         * @return the primary encoding format of the key.
         */
public java.lang.String getFormat() {

             return "X.509";
}
       /**
         * Returns the modulus.
         *
         * @return the modulus
         */
public java.math.BigInteger getModulus() {
             return modulus;
}
       /**
         * Returns the public exponent.
         *
         * @return the public exponent
         */
public java.math.BigInteger getPublicExponent() {
             return publicExponent;

}
       public String toString()
       {
             StringBuffer    buf = new StringBuffer();
             String          nl = System.getProperty("line.separator");

             buf.append("RSA Public Key" + nl);
             buf.append("            modulus: " +
             this.getModulus().toString(16) + nl);
             buf.append("    public exponent: " +
             this.getPublicExponent().toString(16) + nl);

             return buf.toString();
       }
}
```

MsgUtils.Java

```
/**
 * Autor: Bruno
 *
 * Classe que gerencia assinaturas e envelopamentos.
 *
 * Metodos:
 *
 * makeDesede192Key
 * generatePkcs7SignedData
 * verifyPkcs7SignedData
 * getCertificateFromPkcs7SignedData
 * openPkcs7EnvelopedData
 * generatePkcs7EnvelopedData
 *
 *
 *
 */
package pki;

import java.io.ByteArrayInputStream;
import java.security.KeyPair;
import java.security.PrivateKey;
import java.security.SecureRandom;
import java.security.cert.X509Certificate;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import org.bouncycastle.asn1.ASN1InputStream;
import org.bouncycastle.asn1.ASN1Set;
import org.bouncycastle.asn1.cms.ContentInfo;
import org.bouncycastle.asn1.cms.EncryptedContentInfo;
import org.bouncycastle.asn1.cms.OriginatorInfo;
import org.bouncycastle.cms.CMSEnvelopedData;
import org.bouncycastle.cms.CMSEnvelopedDataGenerator;
import org.bouncycastle.cms.CMSProcessable;
import org.bouncycastle.cms.CMSProcessableByteArray;
import org.bouncycastle.cms.CMSSignedData;
import org.bouncycastle.cms.CMSSignedDataGenerator;
import org.bouncycastle.cms.RecipientInformation;
import org.bouncycastle.cms.RecipientInformationStore;
import org.bouncycastle.cms.SignerInformation;
import org.bouncycastle.cms.SignerInformationStore;


public class MsgUtils {

	public static SecureRandom	rand;
	public static KeyGenerator	desede192kg;

	public static final boolean DEBUG = true;

	static {
		try {
			rand = new SecureRandom();

			desede192kg = KeyGenerator.getInstance("DESEDE", "BC");
```

```
                desede192kg.init(168, rand);

            }
            catch(Exception ex) {
                ex.printStackTrace();
            }
        }


        /**
         * Cria uma chave simétrica aleatória.
         */
        public static SecretKey makeDesede192Key() {
            return desede192kg.generateKey();
        }


        private MsgUtils() {
            super();
        }


        /**
         * Gera uma mensagem assinada.
         * @param dados
         * @param origKP
         * @param origCert
         * @return
         * @throws Exception
         */
    public static byte[] generatePkcs7SignedData(byte dados[], KeyPair origKP,
    X509Certificate origCert) throws Exception     {

        ArrayList           certList = new ArrayList();
        CMSProcessable      msg = new CMSProcessableByteArray(dados);//"Hello
        World!".getBytes());

        certList.add(origCert);
        //certList.add(signCert);

        org.bouncycastle.jce.cert.CertStore           certs =
        org.bouncycastle.jce.cert.CertStore.getInstance("Collection",
        new
        org.bouncycastle.jce.cert.CollectionCertStoreParameters(certList),
        "BC");

        CMSSignedDataGenerator gen = new CMSSignedDataGenerator();

        gen.addSigner(origKP.getPrivate(), origCert,
        CMSSignedDataGenerator.DIGEST_SHA1);

        gen.addCertificatesAndCRLs(certs);

        CMSSignedData s = gen.generate(msg, false, "BC");//false nao envia a
        mensagem original no corpo

        ByteArrayInputStream bIn = new ByteArrayInputStream(s.getEncoded());
        ASN1InputStream      aIn = new ASN1InputStream(bIn);

        s = new CMSSignedData(ContentInfo.getInstance(aIn.readObject()));
```

```
        return s.getEncoded();

    }

        /**
         * Verifica uma mensagem asssinada com o certificado passado.
         * @param msg
         * @param dados
         * @param cert
         * @return
         * @throws Exception
         */
public static boolean verifyPkcs7SignedData(byte msg[], byte dados[],
X509Certificate cert) throws Exception   {

ByteArrayInputStream bIn = new ByteArrayInputStream(msg);
ASN1InputStream      aIn = new ASN1InputStream(bIn);

CMSProcessable      msgCMS = new CMSProcessableByteArray(dados);

CMSSignedData s = new CMSSignedData(msgCMS,
ContentInfo.getInstance(aIn.readObject()));

org.bouncycastle.jce.cert.CertStore certs =
s.getCertificatesAndCRLs("Collection", "BC");

SignerInformationStore  signers = s.getSignerInfos();
Collection c = signers.getSigners();
Iterator it = c.iterator();

//verifica soh o primeiro
if (it.hasNext())
{
      SignerInformation   signer = (SignerInformation)it.next();
      Collection          certCollection =
      certs.getCertificates(signer.getSID());

      Iterator        certIt = certCollection.iterator();

      //verifica se deve utilizar o certificado passado por parametro
      if(cert == null){
            cert = (X509Certificate)certIt.next();
      }

      return signer.verify(cert, "BC");
  }

  return false;
}


        /**
         * Recupera o certificado de uma mensagem asssinada.
         * @param msg
         * @param dados
         * @param origKP
         * @param origCert
         * @return
         * @throws Exception
         */
```

```
public static X509Certificate getCertificateFromPkcs7SignedData(byte msg[])
throws Exception  {

      ByteArrayInputStream bIn = new ByteArrayInputStream(msg);
      ASN1InputStream      aIn = new ASN1InputStream(bIn);

      CMSSignedData s = new
      CMSSignedData(ContentInfo.getInstance(aIn.readObject()));

      org.bouncycastle.jce.cert.CertStore certs =
      s.getCertificatesAndCRLs("Collection", "BC");

      SignerInformationStore  signers = s.getSignerInfos();
      Collection c = signers.getSigners();
      Iterator it = c.iterator();

      while (it.hasNext())
      {
            SignerInformation   signer = (SignerInformation)it.next();
            Collection          certCollection =
            certs.getCertificates(signer.getSID());

            Iterator        certIt = certCollection.iterator();
            return (X509Certificate)certIt.next();
      }

        return null;
}


      /**
       * Abre uma mensagem envelopada.
       * @param msg
       * @param key
       * @return
       * @throws Exception
       */
public static byte[] openPkcs7EnvelopedData(byte msg[], PrivateKey key)
throws Exception  {

      CMSEnvelopedData ed = new CMSEnvelopedData(msg);

      RecipientInformationStore  recipients = ed.getRecipientInfos();

      Collection  c = recipients.getRecipients();
      Iterator    it = c.iterator();

      while (it.hasNext())
      {
            RecipientInformation   recipient =
            (RecipientInformation)it.next();

            //recupera apenas para o primeiro destinatario
            return recipient.getContent(key, "BC");

      }

      return null;
}
```

```
        /**
         * gera uma mensagem assinada.
         * @param data
         * @param reciCert
         * @return
         * @throws Exception
         */
public static byte[] generatePkcs7EnvelopedData(byte data[],
X509Certificate reciCert) throws Exception     {

        OriginatorInfo          originatorInfo;
        ASN1Set                 recipientInfos;
        EncryptedContentInfo    encryptedContentInfo;
        ASN1Set                 unprotectedAttrs;

        //SecretKey kek  = makeDesede192Key();

        CMSEnvelopedDataGenerator edGen = new CMSEnvelopedDataGenerator();

        edGen.addKeyTransRecipient(reciCert);

        CMSEnvelopedData ed = edGen.generate(
        new CMSProcessableByteArray(data),
        CMSEnvelopedDataGenerator.DES_EDE3_CBC, "BC");

        //CMSEnvelopedDataGenerator.AES128_CBC, "BC");
        //"1.2.840.113549.3.4", "BC");


        return ed.getEncoded();
    }
}
```

Utils.Java

```
/**
 * Autor: Bruno
 *
 * Contem metodos usados para a implementacao de
 * uma PKI.
 *
 * Metodos:
 *
 * Initialize
 * gravarCertificado
 * gerarKeyPairRSA
 * gravarPrivKeyPkcs12
 * GeneratePBES2Parameters
 * GravarPrivKeyPkcs5
 * gerarCertificadoX509V3
 * gerarCSR
 * getInstanceCertificadoX509
 * getInstanceRequisicaoCertificado
 * lerPrivKeyPkcs12
 * lerPrivKeyPkcs5
 * extrairCN
 * isChavePublicaConfere
 *
 *
 */
package pki;

import org.bouncycastle.jce.*;
import org.bouncycastle.jce.provider.*;
import org.bouncycastle.crypto.params.*;
import org.bouncycastle.crypto.*;
import java.math.*;
import org.bouncycastle.crypto.generators.*;
import java.security.SecureRandom;
import java.security.KeyPair;
import org.bouncycastle.asn1.x509.*;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.KeyStore;
import java.security.Signature;
import java.security.cert.CertificateEncodingException;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import java.io.*;
import org.bouncycastle.util.encoders.*;
import org.bouncycastle.asn1.*;
import java.util.*;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import org.bouncycastle.crypto.paddings.PaddedBufferedBlockCipher;
import org.bouncycastle.crypto.modes.*;
import org.bouncycastle.crypto.engines.*;
import org.bouncycastle.asn1.pkcs.*;


public class Utils {

      private static X509V3CertificateGenerator      CERT_GENERATOR = null;
```

```
	private static JDKX509CertificateFactory CERT_FACTORY = null;
	private static RSAKeyPairGenerator
	RSA_KEYPAIR_GENERATOR=null;
	private static java.text.SimpleDateFormat DATE_FORMAT=new
	java.text.SimpleDateFormat("dd/MM/yyyy");
	private static KeyStore
	PKCS12_KEY_STORE=null;
	private static boolean init=false;


	private Utils() {
		super();
	}

	/**
	 * Inicializa e seleciona provider, gerando par de chaves
	 * criptograficas.
	 */
	private static void initialize() throws java.lang.Exception {

		if(!init){
			//provider
			java.security.Security.addProvider(new
			org.bouncycastle.jce.provider.BouncyCastleProvider());
			CERT_GENERATOR = new X509V3CertificateGenerator();
			CERT_FACTORY= new JDKX509CertificateFactory();
			RSA_KEYPAIR_GENERATOR=new RSAKeyPairGenerator();
			//RSA_KEYPAIR_GENERATOR=KeyPairGenerator.getInstance("RSA
			", "BC");
			PKCS12_KEY_STORE=KeyStore.getInstance("PKCS12", "BC");
			/* adiciona OIDS */
			/* state, or province name - StringType(SIZE(1..64))  */
			final DERObjectIdentifier S = new
			DERObjectIdentifier("2.5.4.4");
			X509Name.OIDLookUp.put(S, "S");
			X509Name.SymbolLookUp.put("S", S);
			/* */
			init=true;
		}
	}

	/**
	 * Grava certificado em disco.
	 * @param cert
	 * @param arquivo
	 * @throws IOException
	 * @throws CertificateEncodingException
	 */
public static void gravarCertificado(X509CertificateObject cert, String
arquivo) throws IOException, CertificateEncodingException {

	File arq = new File(arquivo);

	FileOutputStream fos = new FileOutputStream(arq);

	try{
		fos.write(cert.getEncoded());
	}catch(IOException e){
		throw e;
	}finally{
		//fecha arquivo mesmo se houve excessao
```

```
            fos.close();
      }

  }


/**
 * Gera par de chaves RSA de 1024 bits pubExp = 65535.
 * @return org.bouncycastle.crypto.AsymmetricCipherKeyPair
 * @exception java.lang.Exception The exception description.
 */
public static KeyPair gerarKeyPairRSA() throws java.lang.Exception {

      initialize();

      RSAKeyGenerationParameters algParam=new
      RSAKeyGenerationParameters(new BigInteger("65537", 10), new
      SecureRandom(), 1024, 16);//11 ou 20

      RSA_KEYPAIR_GENERATOR.init(algParam);
      AsymmetricCipherKeyPair par=RSA_KEYPAIR_GENERATOR.generateKeyPair();
      RSAPrivateCrtKeyParameters   priv =
      (RSAPrivateCrtKeyParameters)par.getPrivate();
      RSAKeyParameters             pub  =
      (RSAKeyParameters)par.getPublic();


      //converte
      UtilRSAPublicKey pubKey=new UtilRSAPublicKey(pub.getModulus(),
      pub.getExponent());
      UtilRSAPrivateCrtKey privKey=new UtilRSAPrivateCrtKey( new
      java.security.spec.RSAPrivateCrtKeySpec(
            priv.getModulus(),
            priv.getPublicExponent(),
            priv.getExponent(),
            priv.getP(),
            priv.getQ(),
            priv.getDP(),     //primeExponentP
            priv.getDQ(),     //primeExponentQ
            priv.getQInv()    //crtCoefficient
      ));
      KeyPair kp=new KeyPair(pubKey, privKey);

      //RSA_KEYPAIR_GENERATOR.initialize(1024, new SecureRandom());
      return kp;
}


/**
 * Grava chave privada no formato pkcs12 com senha.
 * @param nomeArq java.lang.String
 * @param senha char[]
 * @exception java.lang.Exception The exception description.
 */
public static void gravarPrivKeyPkcs12(String nomeArq, char[] senha,
PrivateKey privKey, String alias, java.security.cert.X509Certificate cert)
throws java.lang.Exception {

      initialize();

      java.security.cert.X509Certificate chain[]=null;
```

```
        if(cert != null){
              chain=new java.security.cert.X509Certificate[]{cert};
        }
        PKCS12_KEY_STORE.load(null, null);
        PKCS12_KEY_STORE.setKeyEntry(alias, privKey, senha, chain);
        FileOutputStream fos=new FileOutputStream(nomeArq);
        PKCS12_KEY_STORE.store(fos, senha);

        fos.flush();
        fos.close();
}

/**
 * Gera parametros para PKCS5.
 * @return org.bouncycastle.asn1.pkcs.PBES2Parameters
 * @exception Exception       ...
 */

private static PBES2Parameters generatePBES2Parameters() throws Exception {

        byte[] salt=new byte[8];
        int iterationCount=2048;
        byte[] IV=new byte[8];


        SecureRandom random=new SecureRandom();
        random.nextBytes(salt);
        random.nextBytes(IV);

        return generatePBES2Parameters(salt, iterationCount, IV);
}


/**
 * Gera os parametros necessarios a gravacao de um segredo PKCS12.
 *
 * @return org.bouncycastle.asn1.pkcs.PBES2Parameters
 * @param salt                        random password
 * @param iterationCount      iterationCount - 2048 - minimo recomendado eh
1000.
 * @param iv                          Inicialization Vector p/ 3DES
 * @exception Exception       ...
 */
private static PBES2Parameters generatePBES2Parameters(byte[] salt, int
iterationCount, byte[] iv)throws Exception {

        //validacao
        try{
              if(salt.length != 8 || iv.length != 8){
                    throw new Exception();
              }
        }catch(Exception e){
              throw new IllegalArgumentException("Valor do salt ou IV
              inválido. Um OctetString deve ter 8 bytes.");
        }


        //keyDerivationFunc -> PBES2-KDF = KDF2
        DERConstructedSequence keyDerivationFunc=new
        DERConstructedSequence();
        keyDerivationFunc.addObject(PKCSObjectIdentifiers.id_PBKDF2);//oid
```

```
        DERConstructedSequence param=new DERConstructedSequence();
        param.addObject(new DEROctetString(salt));//salt
        param.addObject(new DERInteger(iterationCount));//iterationCount
        keyDerivationFunc.addObject(param);


        //encryptionScheme -> des-EDE3-CBC}
        DERConstructedSequence encryptionScheme=new DERConstructedSequence();
        encryptionScheme.addObject(PKCSObjectIdentifiers.des_EDE3_CBC);
        encryptionScheme.addObject(new DEROctetString(iv));//IV


        //fim
        DERConstructedSequence seq=new DERConstructedSequence();
        seq.addObject(keyDerivationFunc);
        seq.addObject(encryptionScheme);
        PBES2Parameters ret=new PBES2Parameters(seq);
        return ret;
}


        /**
         * Grava uma chave no formato pkcs5 com senha
         * usando des-ede3-cbc.
         * @param nomeArq java.lang.String
         * @param senha char[]
         * @exception java.lang.Exception The exception description.
         */
public static byte[] gravarPrivKeyPkcs5(char[] senha, PrivateKey privKey)
throws java.lang.Exception {

        initialize();

        //monta a chave
        try {
              byte data[] = privKey.getEncoded();
              ByteArrayInputStream bis = new ByteArrayInputStream(data);
              DERInputStream dis = new DERInputStream(bis);
              DERConstructedSequence seq = (DERConstructedSequence)
              dis.readObject();
              PrivateKeyInfo privKeyInfo = new PrivateKeyInfo(seq);

              //cifra
              BufferedBlockCipher cipher =
              new PaddedBufferedBlockCipher(new CBCBlockCipher(new
              DESedeEngine()));
              int keySize = 192;
              PBEParametersGenerator generator = new
              PKCS5S2ParametersGenerator();
              PBES2Parameters alg = generatePBES2Parameters(); //new
              PBES2Parameters(seq2);
              PBKDF2Params func = (PBKDF2Params) alg.getKeyDerivationFunc();
              EncryptionScheme scheme = alg.getEncryptionScheme();

              if (func.getKeyLength() != null) {
                    keySize = func.getKeyLength().intValue() * 8;
              }

              int iterationCount = func.getIterationCount().intValue();
              byte[] salt = func.getSalt();
```

```
        //inicializa generator com a senha passada
        generator.init(
        PBEParametersGenerator.PKCS5PasswordToBytes(senha),
        salt,
        iterationCount);

        CipherParameters param;

        byte[] iv = ((DEROctetString) scheme.getObject()).getOctets();
        param = new
        ParametersWithIV(generator.generateDerivedParameters(keySize),
        iv);

        cipher.init(true, param); //cifragem

        //privateKeyInfo
        ByteArrayOutputStream bout = new ByteArrayOutputStream();
        DEROutputStream dos = new DEROutputStream(bout);
        dos.writeObject(privKeyInfo.getDERObject());
        dos.flush();
        data = bout.toByteArray();

        //byte[]  data = info.getEncryptedData();
        byte[] out = new byte[cipher.getOutputSize(data.length)];
        int len = cipher.processBytes(data, 0, data.length, out, 0);

        try {
              cipher.doFinal(out, len);
        } catch (Exception e) {
              throw new Exception(
              "Erro ao decifrar a chave privada, verifique a senha: " +
              e.getMessage());
        }

        //privkeyInfoCifrado
        DERConstructedSequence encPrivPBE2 = new
        DERConstructedSequence();
        encPrivPBE2.addObject(PKCSObjectIdentifiers.id_PBES2);
        encPrivPBE2.addObject(alg.getDERObject());
        DERConstructedSequence seq2 = new DERConstructedSequence();
        seq2.addObject(encPrivPBE2);
        seq2.addObject(new DEROctetString(out));
        EncryptedPrivateKeyInfo privKeyInfoEnc = new
        EncryptedPrivateKeyInfo(seq2);
        bout = new ByteArrayOutputStream();
        dos = new DEROutputStream(bout);
        dos.writeObject(privKeyInfoEnc.getDERObject());
        dos.flush();
        byte privKeyInfoEncBin[] = bout.toByteArray();

        //base64
        return codificarBase64(privKeyInfoEncBin).getBytes("ISO-8859-
        1");

        } catch (Exception e) {
              throw e;
        }

}
```

```
	/**
	 * Grava uma chave no formato pkcs5 com senha
	 * usando des-ede3-cbc.
	 * @param nomeArq java.lang.String
	 * @param senha char[]
	 * @exception java.lang.Exception The exception description.
	 */
public static void gravarPrivKeyPkcs5(String nmArq, char[] senha,
PrivateKey privKey) throws java.lang.Exception {

	byte out[]=gravarPrivKeyPkcs5(senha, privKey);
	//byte outHex[]=Hex.encode(out);


	FileOutputStream fos=new FileOutputStream(nmArq);
	//fos.write(outHex);
	fos.write(out);
	fos.flush();
	fos.close();

}



/**
 * Gerar certificado autoassinado com um serial number em hexa.
 * EX algoritmo de assinatura: "SHA1withRSAEncryption" ou
"MD5withRSAEncryption"
 * @return java.security.cert.X509Certificate
 * @param subject java.lang.String
 * @param serialNumber java.lang.String
 * @param parChavesRSA org.bouncycastle.crypto.AsymmetricCipherKeyPair
 * @exception java.lang.Exception The exception description.
 */
public static X509CertificateObject gerarCertificadoX509V3(String
proprietario, String emissor, String serialNumber, KeyPair parChavesRSA,
String algoritmoAssinatura) throws java.lang.Exception {

	initialize();

	//gera certificado
	CERT_GENERATOR.reset();

	//dados
	X509Name subj=new X509Name(proprietario);
	X509Name issuer=new X509Name(emissor);
	CERT_GENERATOR.setSubjectDN(subj);
	CERT_GENERATOR.setIssuerDN(issuer);

	//hoje
	Date dt = new Date();
	CERT_GENERATOR.setNotBefore(dt);

	//adiciona 01 ano
	Calendar cal = Calendar.getInstance();
	cal.setTime(dt);
	cal.add(Calendar.YEAR, 1);
	CERT_GENERATOR.setNotAfter(cal.getTime());

	CERT_GENERATOR.setSerialNumber(new BigInteger(serialNumber, 16));
	CERT_GENERATOR.setSignatureAlgorithm(algoritmoAssinatura);
```

```
        CERT_GENERATOR.setPublicKey((PublicKey) parChavesRSA.getPublic());

        //certificado em si
        X509CertificateObject cert=null;
        cert = (X509CertificateObject)
        CERT_GENERATOR.generateX509Certificate(parChavesRSA.getPrivate(),
        "BC");

              return cert;
        }



        /**
         * Gerar CSR.
         * EX algoritmo de assinatura: "MD5withRSA" ou "SHA1withRSA".
         * @return java.security.cert.X509Certificate
         * @param subject java.lang.String
         * @param serialNumber java.lang.String
         * @param parChavesRSA
        org.bouncycastle.crypto.AsymmetricCipherKeyPair
         * @exception java.lang.Exception The exception description.
         */
public static void gerarCSR(String nomeArq, String subject, KeyPair
parChavesRSA, String algAssinatura) throws java.lang.Exception {

        initialize();

        PrivateKey  privKey=parChavesRSA.getPrivate();
        PublicKey   pubKey=parChavesRSA.getPublic();
        X509Name subj = new X509Name(subject);
        PKCS10CertificationRequest req1 =
        new PKCS10CertificationRequest(
        algAssinatura, subj, parChavesRSA.getPublic(), null,
        parChavesRSA.getPrivate());

        byte[] bytes = req1.getEncoded();

        FileOutputStream fw=new FileOutputStream(nomeArq);
        fw.write(bytes);
        fw.flush();
        fw.close();

}

/**
 * Obtem uma instancia de um certificado X509.
 * param nomeArq java.lang.String
 * @return org.bouncycastle.jce.provider.X509CertificateObject
 */
public static X509CertificateObject getInstanceCertificadoX509(String
nomeArq) throws Exception {


        initialize();

        X509CertificateObject cert = null;
        FileInputStream fis= null;

        try{
              fis=new FileInputStream(nomeArq);
```

```
            cert=(X509CertificateObject)CERT_FACTORY.engineGenerateCertific
            ate(fis) ;
                  fis.close();
            }catch(Exception e){
                  throw e;
            }

            return cert;
      }

/**
 * Gera requesicao de certificado.
 * @param nomeArq
 * @return
 * @throws Exception
 */
public static PKCS10CertificationRequest
getInstanceRequisicaoCertificado(String nomeArq) throws Exception {

      initialize();

      PKCS10CertificationRequest req = null;
      FileInputStream fis= null;

      byte[] dados;
      fis = new FileInputStream(nomeArq);

      dados = new byte[fis.available()];
      fis.read(dados);

      fis.close();

      return new PKCS10CertificationRequest(dados);
}

/**
 * Insira a descrição do método aqui.
 * @return byte[]
 * @param dados byte[]
 * @exception java.lang.Exception A descrição da exceção.
 */
public static String codificarBase64(byte[] dados) throws
java.lang.Exception {
      /* Codificador de base64 */
      String ret=null;
      try{
            Base64Encoder base64Encoder=new Base64Encoder( new
            String(dados, "ISO-8859-1") );
            ret=base64Encoder.processString();
      }catch(Exception e){
            throw new Exception(e.getMessage());
      }

      return ret;
}
/**
 * Insira a descrição do método aqui.
 * @return byte[]
 * @param dados byte[]
 * @exception java.lang.Exception A descrição da exceção.
 */
```

```
public static byte[] decodificarBase64(String dados) throws
java.lang.Exception {
	/* Codificador de base64 */
	byte ret[]=null;
	try{
		ret=new Base64Decoder(dados).processString().getBytes("ISO-
		8859-1");
	}catch(Exception e){
		throw new Exception(e.getMessage());
	}

	return ret;
}

/**
 * Le um certificado no formato pkcs12 com senha.
 * @param nomeArq java.lang.String
 * @param senha char[]
 * @exception java.lang.Exception The exception description.
 */
public static PrivateKey lerPrivKeyPkcs12(String nomeArq, char[] senha,
String alias) throws java.lang.Exception {

	initialize();

	FileInputStream fis=new FileInputStream(nomeArq);
	PKCS12_KEY_STORE.load(fis, senha);

	PrivateKey privKey=null;
	if(!PKCS12_KEY_STORE.containsAlias(alias)){
		throw new IllegalArgumentException("Alias nao existe: "+alias);
	}
	privKey=(PrivateKey)PKCS12_KEY_STORE.getKey(alias, senha);

	return privKey;
}

/**
 * Le um chave no formato pkcs5 com senha
 * usando des-ede3-cbc.
 * @param nomeArq java.lang.String
 * @param senha char[]
 * @exception java.lang.Exception The exception description.
 */
public static PrivateKey lerPrivKeyPkcs5(byte[] dados, char[] senha)
	throws java.lang.Exception {

	initialize();

	DERConstructedSequence seq;

	//decifra
	BufferedBlockCipher cipher =
	new PaddedBufferedBlockCipher(new CBCBlockCipher(new
	DESedeEngine()));
	int keySize = 192;
	PBEParametersGenerator generator = new PKCS5S2ParametersGenerator();
	ByteArrayInputStream bIn = new ByteArrayInputStream(dados);
	DERInputStream dIn = new DERInputStream(bIn);
	EncryptedPrivateKeyInfo info;
```

```
//obtem a chave privada encriptada
try {
      info = new EncryptedPrivateKeyInfo((DERConstructedSequence)
      dIn.readObject());
} catch (Exception e) {
throw new Exception("Formato de chave invalido: " + e.getMessage());
}

DERConstructedSequence algParam =
(DERConstructedSequence)
info.getEncryptionAlgorithm().getParameters();

PBES2Parameters alg = new PBES2Parameters(algParam);
PBKDF2Params func = (PBKDF2Params) alg.getKeyDerivationFunc();
EncryptionScheme scheme = alg.getEncryptionScheme();

if (func.getKeyLength() != null) {
      keySize = func.getKeyLength().intValue() * 8;
}

int iterationCount = func.getIterationCount().intValue();
byte[] salt = func.getSalt();

//inicializa generator com a senha passada
generator.init(
      PBEParametersGenerator.PKCS5PasswordToBytes(senha),
      salt,
      iterationCount);

CipherParameters param;

byte[] iv = ((DEROctetString) scheme.getObject()).getOctets();

param = new
ParametersWithIV(generator.generateDerivedParameters(keySize), iv);

cipher.init(false, param);

byte[] data = info.getEncryptedData();
byte[] out = new byte[cipher.getOutputSize(data.length)];
int len = cipher.processBytes(data, 0, data.length, out, 0);

try {
      cipher.doFinal(out, len);
} catch (Exception e) {
      throw new Exception(
      "Erro ao decifrar a chave privada, verifique a senha: " +
      e.getMessage());
}

//monta a chave
      ByteArrayInputStream bis = new ByteArrayInputStream(out);
      DERInputStream dis = new DERInputStream(bis);
      try {
            seq = (DERConstructedSequence) dis.readObject();
      } catch (IOException e) {
            throw e;
      }
      PrivateKeyInfo privKeyInfo = new PrivateKeyInfo(seq);
      UtilRSAPrivateCrtKey privKey = new
      UtilRSAPrivateCrtKey(privKeyInfo);
```

```
      return privKey;
}
/**
 * Le um chave no formato pkcs5 com senha
 * usando des-ede3-cbc.
 * @param nomeArq java.lang.String
 * @param senha char[]
 * @exception java.lang.Exception The exception description.
 */
public static PrivateKey lerPrivKeyPkcs5(String nomeArq, char[] senha)
throws java.lang.Exception {

      //le arquivo
      FileInputStream fis=new FileInputStream(nomeArq);
      byte dados[]=new byte[fis.available()];
      fis.read(dados);


      dados=decodificarBase64(new String(dados, "ISO-8859-1"));
      return lerPrivKeyPkcs5(dados, senha);
}

/**
 * Extrai o common name do subject.
 * @param name
 * @return
 */
public static String extrairCN(String name){

      int i = name.indexOf("CN=");
      if(i >= 0){
            //inicio do nome
            i = i + 3;

            //procura virgula
            int f = name.indexOf(",", i);
            if(f  < 0){
                  f = name.length();
            }

            name = name.substring(i, f);

      }

      return name;
}

/**
 * Verifica se uma chave publica bate com a privada.
 * @param privKey
 * @param pubKey
 * @return
 */
public static boolean isChavePublicaConfere(RSAPrivateKey privKey,
RSAPublicKey pubKey){

try{
      if(   pubKey.getModulus().equals(privKey.getModulus())){

                  //gera uma assinatura para teste
```

```
Signature assinatura = Signature.getInstance("SHA1withRSA");
assinatura.initSign(privKey);
assinatura.update( (byte)0x1);
byte dadosAsinados[] = assinatura.sign();

//verifica assinatura
assinatura.initVerify(pubKey);
assinatura.update( (byte)0x1);
return assinatura.verify(dadosAsinados);
}

}catch(Exception e){
   }

       return false;
  }

}
```

FileUtils.Java

```
/**
 * Autor: Bruno
 *
 * Executa a copia de arquivos para os diretorios
 * definidos.
 *
 */
package util;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class FileUtils {


	private FileUtils() {
		super();
	}


	/**
	 * Copia arquivo.
	 *
	 */
	public static void copiarArquivo(File origem, File destino) throws
	IOException {

		if(origem.exists() && origem.isFile()){

			byte buffer[] = new byte[10000];

			FileInputStream fis = new FileInputStream(origem);
			FileOutputStream fos = new FileOutputStream(destino);

			int count;
			while( (count = fis.read(buffer)) > 0){
				fos.write(buffer, 0, count);
			}

			fis.close();
			fos.close();

		}else{
		throw new IOException("O arquivo de origem é inválido: " +
		origem.getAbsolutePath());
		}
	}

}
```

GuiUtils.java

```
/**
 * Autor: Bruno
 *
 * Classe que gerencia telas para salvar e abrir arquivos.
 *
 */
package util;

import java.awt.Component;
import java.io.File;
import javax.swing.JFileChooser;

public class GuiUtils {

	private GuiUtils() {
		super();
	}

	/**
	 * Pergunta ao usuário um nome de arquivo para abrir.
	 *
	 */
	public static File promptFileForOpen(Component parent, String titulo,
	String dirAtual) {

		JFileChooser fc = new JFileChooser(dirAtual);
		fc.setDialogTitle(titulo);

		int resp = fc.showOpenDialog(parent);
		if(resp == JFileChooser.APPROVE_OPTION){
			return fc.getSelectedFile();
		}

		return null;
	}

	/**
	 * Pergunta ao usuario um nome de arquivo para salvar.
	 *
	 */
		public static File promptFileForSave(Component parent, String
	titulo, String dirAtual) {

		JFileChooser fc = new JFileChooser(dirAtual);

		fc.setDialogTitle(titulo);

		int resp = fc.showSaveDialog(parent);

		if(resp == JFileChooser.APPROVE_OPTION){

			return fc.getSelectedFile();
		}

		return null;
	}

}
```