UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

PROTÓTIPO DE SOFTWARE PARA EMISSÃO DE CERTIFICADOS DIGITAIS PARA OBJETOS DISTRIBUÍDOS

DERLEI ALVARO MATHIAS

BLUMENAU
2007

2007/1-08

DERLEI ALVARO MATHIAS

PROTÓTIPO DE SOFTWARE PARA EMISSÃO DE CERTIFICADOS DIGITAIS PARA OBJETOS DISTRIBUÍDOS

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Paulo Fernando da Silva, Titulação - Orientador

BLUMENAU
2007

2007/1-08

PROTÓTIPO DE SOFTWARE PARA EMISSÃO DE CERTIFICADOS DIGITAIS PARA OBJETOS DISTRIBUÍDOS

Por

DERLEI ALVARO MATHIAS

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Paulo Fernando da Silva, Titulação – Orientador, FURB
Membro:	Prof. Francisco Adell Péricas, Titulação – FURB
Membro:	Prof. Sérgio Stringari, Titulação – FURB

Blumenau, 10 de julho de 2007

Dedico este trabalho a minha noiva Fabiana Bankhardt, pelo apoio recebido durante estes anos de graduação.

AGRADECIMENTOS

Primeiramente a Deus, por ter me dado condições de realizar este trabalho.

Ao meu pai, Alvaro Mathias pela ajuda financeira recebida, possibilitando com isso mais uma conquista em minha vida.

Aos colegas e professores do curso de Ciências da computação pela troca de conhecimentos e experiências.

Aos meus amigos, pela compreensão quanto às horas ausente.

Ao meu orientador, Paulo Fernando da Silva, pela orientação e apoio no desenvolvimento deste trabalho.

Ao meu grande amigo, Jean Carlos Pereira pela ajuda no desenvolvimento deste trabalho.

Algo é só impossível até que alguém duvide e acabe provando o contrário.

Albert Einstein

RESUMO

Este trabalho apresenta o desenvolvimento de um protótipo de software capaz de emitir certificados digitais auto-assinados, emitir e revogar certificados digitais para os objetos distribuídos e, juntamente com a revogação, mantém uma lista dos certificados revogados. No desenvolvimento do protótipo foram utilizadas técnicas de criptografia como o algoritmo RSA responsável pela emissão do par de chaves e o algoritmo MD5 utilizado para gerar *hash code,* podendo assim assinar o certificado digitalmente.

Palavras-chave: Certificado digital. Comunicação segura. Criptografia.

ABSTRACT

Abstract This work presents the development of an prototype of software capable to emit auto-signed digital certificates, to emit and to revoke digital certificates for objects distributed and together with the revocation keeps a list of the revoked certificates. In the development of the prototype criptography techniques had been used as responsible algorithm RSA for the emission of the pair of keys and used algorithm MD5 to generate hash code, thus being able to sign the certificate digitally.

Key- words: Digital certificate. Safe communication. Criptography

LISTA DE ILUSTRAÇÕES

Figura 1 – Cenário de autenticação 21
Figura 2 – Cenário de Autenticidade 22
Figura 3 – Diretório público 23
Figura 4 – Cenário distribuição de chave pública. 24
Figura 5 – Cenário Certificado de Chave Pública 24
Figura 6 – Classes certificação digital 30
Figura 7 – Diagrama classe estudo de caso 34
Figura 8 – Iniciando *CAServer* 35
Figura 9 – Iniciado *TransmitServer* 36
Figura 10 – Inicia *TransmitClient* 37
Quadro 1 – Emissão certificado raiz 38
Quadro 2 – Emissão par de chaves 39
Quadro 3 – Grava certificado 39
Quadro 4 – Emissão certificado objeto 40
Quadro 5 – Registro Servidor *CAServer* 40
Quadro 6 – Método Registrar 41
Quadro 7 – Valida ou invalida certificado 41
Quadro 8 – Referência servidor CA 42
Quadro 9 – Funcionamento do protótipo 42
Quadro 10 – Inicia *CAServer* 43
Quadro 11 – Certificado da AC geral 44
Quadro 12 – Certificado digital detalhes 44
Quadro 13 – Inicia *TransmitServer* 45
Quadro 14 – Certificado digital *TransmitServer* geral 45
Quadro 15 – Certificado digital *TramistServer* detalhes 46
Quadro 16 – Inicia *TramistClient* 46
Quadro 17 – Certificado digital cliente geral 47
Quadro 18 – Certificado digital cliente detalhes 48
Quadro 19 – Troca mensagens cliente 49
Quadro 20 – Troca mensagens servidor 49

Quadro 21 – Lista de certificados revogados 50
Quadro 22 – Servidor com certificado vencido..51

LISTA DE SIGLAS

AC – Autoridade Certificadora

API – *Application Programming Interface*

AR – Autoridade de Registro

BCC – Curso de Ciências da Computação – Bacharelado

DSC – Departamento de Sistemas e Computação

ITU-T – *International Telecommunication Union Standardization*

Java RMI – Java *Remote Method Invocation*

JVM – Java Virtual Machine

LCR – Lista de Certificados Revogados

MD5 – *Message Digest* 5

RSA – *Rivest-Shamir-Adelman*

UML – *Unifild Modeling Language*

SUMÁRIO

1 INTRODUÇÃO 13

1.1 OBJETIVOS DO TRABALHO 14

1.2 ESTRUTURA DO TRABALHO 14

2 FUNDAMENTAÇÃO TEÓRICA 15

2.1 SISTEMAS DISTRIBUÍDOS 15

2.1.1 Compartilhamento de Recursos 16

2.1.2 Abertura 16

2.1.3 Concorrência 16

2.1.4 Escalabilidade 17

2.1.5 Tolerância a Falhas 17

2.1.6 Transparência 17

2.1.7 Objeto Distribuído 18

2.1.8 Java RMI 18

2.2 SEGURANÇA 19

2.2.1 Criptografia de Chaves Públicas 20

2.2.2 Distribuição de Chaves 22

2.2.3 Certificados Digitais 25

2.2.4 Certificados X.509 26

3 DESENVOLVIMENTO DO TRABALHO 29

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO 29

3.2 ESPECIFICAÇÃO 30

3.2.1 Diagrama de Classe 30

3.2.1.1 Classe `Raiz` 30

3.2.1.2 Classe `CertificateObj` 31

3.2.1.3 Classe CA 32

3.2.1.4 Classe `CAServer` 32

3.2.1.5 Classe `ObjetoCertificado` 33

3.2.2 Especificação do estudo de caso 33

3.2.3 Diagrama de seqüência 34

3.3 IMPLEMENTAÇÃO 37

3.3.1 Técnicas e ferramentas utilizadas 38

3.3.2 Operacionalidade da implementação 42
3.4 RESULTADOS E DISCUSSÃO 50
4 CONCLUSÕES 52
4.1 EXTENSÕES 52
REFERÊNCIAS BIBLIOGRÁFICAS 54

1 INTRODUÇÃO

Segundo Parra (2002, p. 11), garantir a segurança no envio e recebimento de mensagens não é necessidade da atualidade. Desde os tempos primitivos, a preocupação com a segurança era constante, e garantir a confidencialidade de uma informação era de suma importância. Com o aumento do poder computacional e dos meios de comunicação, os métodos usados para assegurar o conteúdo das mensagens também evoluíram na mesma proporção.

Uma forma de garantir a troca de mensagens segura entre objetos distribuídos é com a utilização de certificados digitais. Para Albuquerque (2001, p. 360), um objeto é considerado distribuído se estiver em outra máquina virtual que pode ou não, estar em outro computador. Por sua vez, um certificado digital é como uma carteira de identidade que identifica o objeto na rede, de maneira que somente o objeto para o qual se destina a mensagem poderá ter acesso ao conteúdo. Mas para que se possa confiar na veracidade de um certificado digital necessita-se saber quem o emitiu, conhecendo assim a forma como foi gerado (BATTISTI, 2003). Neste sentido, uma Autoridade Certificadora (AC) é responsável desde a emissão até a revogação de um certificado digital.

Um certificado, para que possa ser usado amplamente por várias aplicações, necessita seguir um padrão, como por exemplo o X.509 (STALLINGS, 2003, p. 420).

Diante do exposto, neste trabalho foi desenvolvido um *middleware* com funcionalidades de uma AC, onde o certificado digital primeiramente é auto-assinado para então poder assinar os certificados para os objetos remotos.

Com o desenvolvimento deste aplicativo foi possível atender às requisições dos objetos distribuídos, gerenciar a emissão, a validade e a revogação dos certificados digitais, permitindo assim a comunicação entre os objetos com certificação digital. A comunicação entre objetos distribuídos foi especificada e implementada apenas para testar à funcionalidade do *middleware*, desta forma a comunicação entre os objetos remotos não faz parte dos objetivos deste protótipo.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho foi desenvolver um *middleware* capaz de emitir certificados digitais para objetos distribuídos, permitindo que estes objetos realizem uma comunicação utilizando seus certificados.

Os objetivos específicos do trabalho são:

a) gerar certificado digital auto-assinado seguindo o padrão X.509;
b) gerar certificado digital para objetos remotos seguindo o padrão X.509;
c) revogar certificados digitais dos objetos remotos;
d) gerar lista de certificados revogados.

1.2 ESTRUTURA DO TRABALHO

No capítulo 2 são apresentados aspectos teóricos que fundamentam o desenvolvimento deste *middleware*. O capítulo 3 descreve a especificação e o desenvolvimento. Já o capítulo 4, apresenta a conclusão do presente trabalho, assim como sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados aspectos teóricos relacionados ao trabalho. Essa apresentação está dividida nas seguintes seções: sistemas distribuídos, objetos distribuídos, Java RMI, chaves públicas, distribuição de chaves, certificados digitais, certificados X.509.

2.1 SISTEMAS DISTRIBUÍDOS

Para Tanenbaum e Steen (2002, p. 02), um sistema distribuído é uma coleção de computadores independentes que parecem como um único sistema para o usuário.

Um sistema é distribuído quando constitui-se por um conjunto de processadores autônomos conectados através de um subsistema de comunicação, que cooperam através da troca de mensagens. Esse tipo de sistema deve apresentar duas características inerentes: a transparência na sua utilização e o alto grau de tolerância à falhas (CASETTI; AKAMATU; KIRNER, 1993).

Conforme Tanenbaum e Steen (2002, p 02), o desenvolvimento de sistemas distribuídos tornou-se possível com o surgimento das redes locais de computadores no início da década de 70. Essas redes permitiram que, computadores que antes trabalhavam de maneira isolada, pudessem ser interligados, trazendo vários benefícios para a computação. Mais recentemente, a disponibilidade de computadores pessoais e estações de trabalho de maior desempenho possibilitou que sistemas distribuídos fossem implementados com grandes vantagens em relação aos sistemas centralizados, em termos de desempenho, disponibilidade e custos.

Atualmente, segundo Casetti, Akamatu, Kirner (1993), os sistemas distribuídos estão sendo pesquisados, desenvolvidos e apresentados como alternativa aos grandes sistemas centralizados. Existem quatro vantagens básicas de se projetar uma aplicação para um sistema distribuído:

a) diminuição do tempo de execução da aplicação;
b) aumento do grau de confiabilidade e disponibilidade da aplicação;
c) o uso de partes do sistema para fornecer especialização funcional;
d) a inerente distribuição da aplicação.

Segundo Hoppe (2005, p. 34), um sistema distribuído apresenta as seguintes características: compartilhamento de recursos, abertura, concorrência, escalabilidade, tolerância à falhas e transparência.

2.1.1 Compartilhamento de Recursos

Conforme Hoppe (2005, p. 37), o compartilhamento de recursos permite a diminuição de custos, compartilhamento de dados e o trabalho cooperativo. Caracteriza uma necessidade já encontrada em sistemas centralizados. Um recurso em um sistema distribuído é, geralmente, armazenado em um único computador. O gerenciamento desses recursos é efetuado por uma entidade chamada administrador de recursos, que possibilita o uso de algum mecanismo de comunicação, para realizar o acesso ao recurso desejado em uma máquina remota.

2.1.2 Abertura

Para Hoppe (2005, p. 36), um sistema computacional é caracterizado como aberto se novos recursos podem ser adicionados ao sistema sem causar prejuízos aos módulos já existentes. Para que isso seja possível, as interfaces para acesso aos recursos devem ser bem definidas, de modo que a inclusão de um novo recurso ao sistema ocorra de maneira natural.

2.1.3 Concorrência

Hoppe (2005, p. 36), afirma que a concorrência e a execução paralela surgem naturalmente em sistemas distribuídos, devido à separação das atividades dos usuários, independência de recursos e a localização de processos servidores em computadores separados.

2.1.4 Escalabilidade

Para Hoppe (2005, p. 37), a possibilidade de ampliação de um sistema distribuído, com relação a sua capacidade computacional, deve ser considerada. Essa ampliação representa uma necessidade gerada por vários fatores como por exemplo, inclusão de novos usuários com a sobrecarga em algum servidor, e deve ser feita de modo a não afetar a utilização do sistema.

2.1.5 Tolerância a Falhas

Segundo Hoppe (2005, p. 37), a tolerância a falhas é uma característica importante, pois muitas vezes os computadores falham, gerando resultados incorretos ou dados inconsistentes. A tolerância a falhas em um sistema distribuído pode ser alcançada através de dois mecanismos: a redundância de *hardware* e redundância de *software*.

2.1.6 Transparência

Conforme Tanenbaum e Steen (2002, p. 5), o sistema oculta do usuário e de suas aplicações a existência de serviços distribuídos, fazendo o usuário pensar que só exista um sistema. As principais características são as seguintes:

a) acesso: oculta diferenças na representação de dados e como um recurso é alcançado;
b) localização: oculta onde um recurso está localizado;
c) migração: oculta a movimentação de um recurso para outra posição;
d) relocação: oculta a movimentação de um recurso para outra posição enquanto está sendo usado;
e) replicação: oculta a distribuição de um recurso em vários servidores;
f) concorrência: oculta o recurso compartilhado por vários usuários;
g) falha: oculta a falta e recuperação de um recurso;
h) persistência: oculta quando um recurso *software* está em memória ou em disco.

2.1.7 Objeto Distribuído

Conforme Montez (1998), um objeto distribuído é essencialmente um componente, uma peça de *software* com inteligência auto-contida, que pode interoperar com outros objetos distribuídos através de sistemas operacionais, redes, linguagens, aplicações, ferramentas e equipamentos diversos. Os objetos distribuídos possuem as mesmas características principais dos objetos das linguagens de programação: encapsulamento, polimorfismo e herança, tendo, dessa forma, as mesmas principais vantagens: fácil reusabilidade, manutenção e depuração, só para citar algumas. Um outro benefício da utilização de objetos distribuídos, aplicado especificamente em sistemas de tempo real, é o polimorfismo de desempenho ou polimorfismo temporal. Este mecanismo permite que uma interface possua várias implementações diferentes do mesmo método, cada uma com um tempo de execução diferente. Dependendo de determinados aspectos temporais, o método que puder atender aos requisitos temporais especificados pelo cliente naquele momento, será executado.

Segundo Montez (1998), um objeto distribuído não opera sozinho, a princípio ele é construído para trabalhar com outros objetos e, para isso, precisa de uma espécie de barramento. Estes barramentos fornecem infra-estrutura para os objetos, adicionando novos serviços que podem ser herdados durante a construção do objeto, ou mesmo em tempo de execução para alcançar altos níveis de colaboração com outros objetos independentes.

2.1.8 Java RMI

A tecnologia Java RMI, possibilita a comunicação entre objetos de diferentes máquinas virtuais e tem como objetivo integrar a linguagem de programação Java ao modelo de computação baseado em objetos distribuídos. Programas RMI apresentam uma arquitetura cliente-servidor onde os clientes invocam métodos que atuam sobre objetos nos servidores. Em programas RMI, um método é executado na máquina onde se encontra o objeto sobre o qual atua. Este objeto pode estar no mesmo ou em outro computador. Invocando um método sobre um objeto remoto, o cliente RMI atua sobre um objeto local, fazendo-se passar pelo objeto remoto. Este objeto local é chamado de *stub*. O *stub* age como um proxy do objeto remoto e esconde do cliente o uso dos serviços providos pelo protocolo de transporte (ALBUQUERQUE, 2001, p. 360).

Conforme Hoppe (2005, p. 47), os clientes RMI interagem com objetos remotos através de interfaces remotas bem definidas, nunca diretamente com as classes que implementam as interfaces. O modelo RMI utiliza a serialização de objetos para converter objetos em *streams* de *bytes* para a transmissão.

Segundo a Sun (2007), Java RMI fornece um modelo simples e direto para a computação distribuída com os objetos Java. Estes objetos podem ser objetos novos, ou podem ser envoltórios simples Java em torno de um API existente. Esta tecnologia tem como base as seguintes características:

a) orientado a objeto: pode-se passar objetos inteiros com argumentos e valores do retorno;
b) comportamento móvel: move as execuções da classes do cliente para o usuário e o usuário para o cliente;
c) seguro: utiliza os mecanismos internos de segurança do Java, permitindo assim que o sistema seja seguro;
d) fácil de implementar: simples de implementar usuários remotos e clientes que alcançam aqueles usuários. Uma relação remota é uma relação real de Java.
e) conecta os sistemas: interage com os sistemas existentes através da relação nativa JNI do método Java. Com o RMI e o JNI podem-se escrever clientes em Java e usar a execução existente do usuário;
f) portável: o sistema 100% portátil a qualquer JVM;
g) coleta distribuída do lixo: utiliza a característica distribuída para coleta do lixo dos objetos remotos dos usuários que já não são mais referenciados por clientes da rede;
h) computação paralela: permite aos usuários explorar linhas de Java para o processamento simultâneo.

2.2 SEGURANÇA

Conforme Silva (2004, p.23), a segurança refere-se à proteção existente sobre as informações de empresas ou pessoas, isto é, segurança tanto das informações corporativas quanto das pessoais. A segurança de uma determinada informação pode ser afetada por fatores comportamentais, como por exemplo, o ambiente ou a infra-estrutura que a cerca,

tendo como objetivo furtar, destruir ou modificar as informações. Para garantir a segurança necessita-se atender três premissas como confidencialidade, integridade e disponibilidade:

a) confidencialidade: limita o acesso a informação às entidades legítimas ou a que o proprietário da informação autorizar;
b) integridade: as informações não podem ser modificadas em trânsito na rede;
c) disponibilidade: garante que as informações sempre estejam disponível para o uso legítimo.

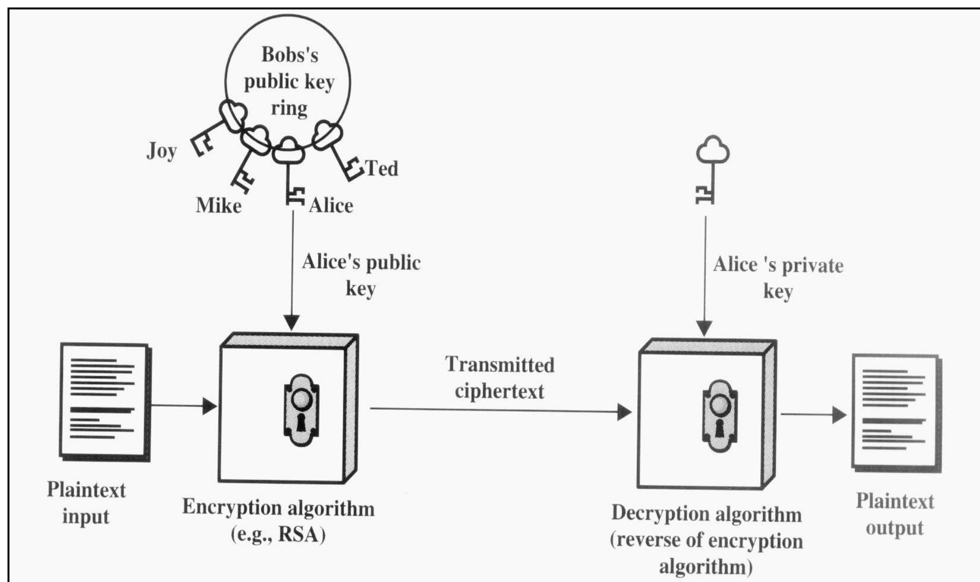
Para atender a estes atributos pode-se utilizar de diversas técnicas dentre as quais criptografia de chaves públicas, distribuição de chaves, certificados digitais e certificados X.509.

2.2.1 Criptografia de Chaves Públicas

Segundo a CertiSign (2006), a criptografia de chave pública resolve várias deficiências da criptografia de chave simétrica, como por exemplo o compartilhamento e chaves. Na criptografia de chave pública, um indivíduo, ou organização, possui duas chaves, uma chave pública e outra chave privativa. Conforme o Centro de Estudos, Respostas e Tratamento de Incidentes de Segurança no Brasil (2005), a criptografia de chaves pública utiliza duas chaves distintas, uma para codificar e outra para decodificar mensagens. Neste método cada entidade mantém duas chaves, uma pública que pode ser divulgada livremente e outra privada, que deve ser mantida em segredo pelo seu dono.

No modelo de confidencialidade têm-se dois usuários, Bob e Alice, para se comunicar de maneira sigilosa, adotam-se os seguintes procedimentos:

a) Bob codifica uma mensagem utilizando a chave pública de Alice, que está disponível para uso de qualquer objeto;
b) depois de criptografada, Bob envia a mensagem para Alice;
c) Alice recebe e decodifica a mensagem, utilizando sua chave secreta, que é apenas de seu conhecimento;
d) se Alice quiser responder a mensagem ou enviar outra mensagem, deverá realizar o mesmo procedimento, mas utilizando a chave pública Bob.

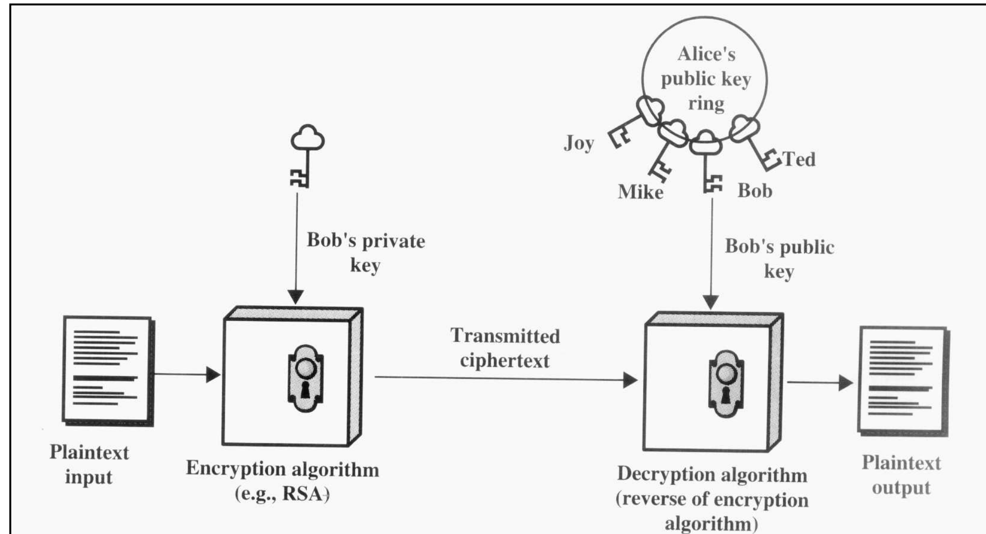


Fonte: Stallings (2003, p. 261).

Figura 1 – Cenário de autenticação

No modelo de autenticidade têm-se dois usuários, Bob e Alice, para se comunicar de maneira autêntica, adota-se os seguintes procedimentos:

a) Bob codifica uma mensagem utilizando a sua chave secreta, que é apenas de seu conhecimento;
b) depois de criptografada, Bob envia a mensagem para Alice;
c) Alice recebe e decodifica a mensagem utilizando a chave pública de Bob;
d) se Alice quiser responder a mensagem ou enviar outra mensagem, deverá realizar o mesmo procedimento, mas utilizando a sua chave secreta.



Fonte: Stallings (2003, p. 261)

Figura 2 – Cenário de Autenticidade

A desvantagem das chaves assimétricas para as simétricas está na velocidade de processamento, pois as chaves assimétricas são bem mais lentas que as simétricas, isso porque as chaves simétricas utilizam a mesma chave para encriptar e decriptar o texto.

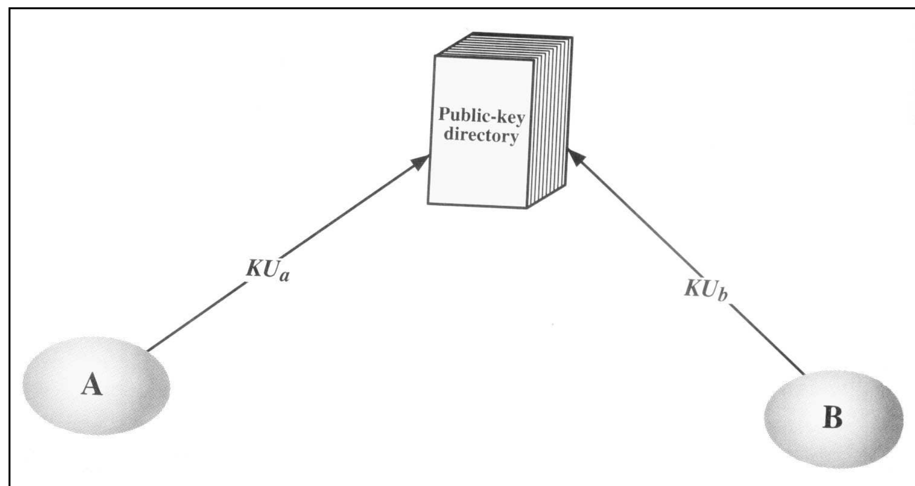
2.2.2 Distribuição de Chaves

Conforme Stallings (2003, p. 286), diversas técnicas foram propostas para a distribuição de chaves públicas, dentre as quais pode-se destacar: anúncio público, diretório público, autoridade de chave pública e certificados de chave pública.

No anúncio público as chaves são anunciadas publicamente, assim se houver algum algoritmo amplamente aceito de chave pública, como o *RSA*, o usuário pode emitir sua chave pública a qualquer outro participante. Uma lista mantida na internet, embora conveniente, tem uma fraqueza principal: qualquer um pode forjar um anúncio público isto é, algum falsário poderia fingir ser o usuário A e emitir uma chave pública a um outro participante. Até a hora que o usuário A descobre o falsário, e alerta o outro participante, o mesmo pode ler todas as mensagens cifradas pretendidas para o usuário A.

No diretório público um grau de segurança pode ser conseguido mantendo um diretório dinâmico de chaves públicas. A manutenção e a distribuição deste diretório necessitam ser de responsabilidade de alguma entidade ou organização de confiança. Para isso inclui-se os seguintes elementos como, mantendo um diretório com a chave de entrada para

cada usuário, onde cada usuário registra uma chave pública com a autoridade do diretório, e este é periodicamente atualizado. A figura 3 ilustra o diretório público:



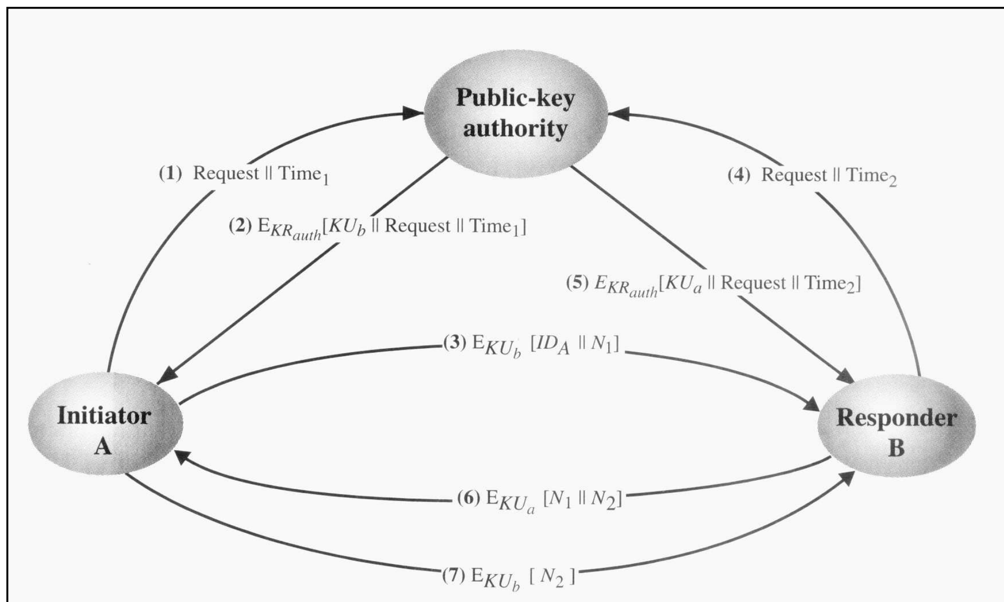
Stallings (2003, p. 288)

Figura 3 – Diretório público

Uma maior segurança para a distribuição chave pública pode ser obtido fornecendo um controle mais rígido sobre a distribuição das chaves. Um cenário da distribuição de chaves é representado na figura 4:

a) "A" envia uma requisição de estabelecimento de comunicação com B para a autoridade de chave pública ($Request||Time_1$);
b) autoridade responde a mensagem encriptada com sua chave privada, contendo a chave pública de "B" ($E_{KRauth}[KU_B|| Request||Time_1]$);
c) "A" envia a sua identificação com N_1 para "B" ($KU_b[ID_A||N_1]$);
d) "B" requer a chave pública de "A" ($Request||Time_2$);
e) "B" recebe a mensagem da autoridade ($E_{KRauth}[KU_A|| Request||Time_2]$);
f) "B" envia uma mensagem para "A", encriptada com a chave pública de "A". Esta mensagem contem N_1, que somente "B" poderia decifrar (passo 3) e outro N_2 ($KU_A[N_1|| N_2]$);
g) "A" retorna N_2 encriptado com a chave pública de B ($KU_B[N_2]$).

N_1 e N_2 são valores randômicos gerados pelo emissor e que devem estar contidos na mensagem de resposta do receptor.

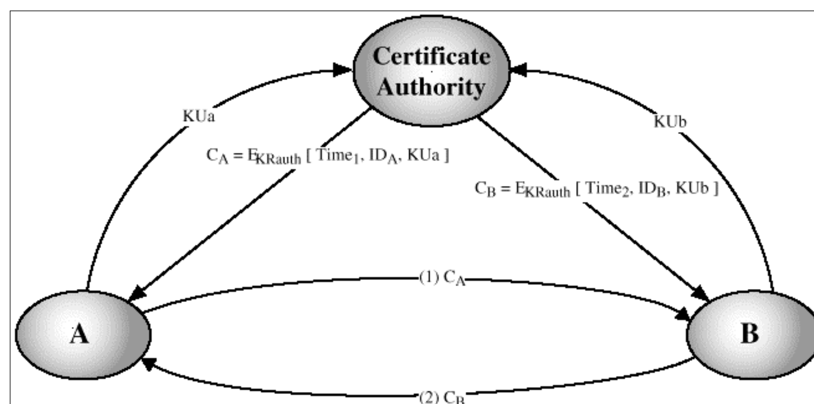


Fonte: Stallings (2003, p.290).

Figura 4 – Cenário distribuição de chave pública.

Um cenário da solicitação de um certificado digital e comunicação entre objetos é representado na figura 5, sendo que :

a) "A" envia sua chave pública para a autoridade certificadora (KU_A);
b) "A" recebe da autoridade uma mensagem encriptada com a chave privada da autoridade que contém a chave pública "A", a identificação de "A" e a validade do certificado ($C_A = E_{KRauth}[Time_1||ID_A||KU_A]$);
c) "B" envia sua chave pública para a autoridade certificadora (KU_B);
d) "B" recebe da autoridade uma mensagem encriptada com a chave privada da autoridade que contém a chave pública "B", a identificação de "B" e a validade do certificado ($C_B=E_{KRauth}[Time_2||ID_B||KU_B]$).



Fonte: Stallings (2003, p. 290).

Figura 5 – Cenário Certificado de Chave Pública

2.2.3 Certificados Digitais

Conforme Silva (2004, p. 139), um certificado é a junção entre uma entidade e uma chave pública. A chave pública, a entidade e algumas informações únicas são colocadas dentro de um documento digital. Este documento digital é chamado de certificado, e para garantir a legitimidade do documento, uma entidade de confiança assina digitalmente o certificado, garantindo com isso o seu conteúdo. Esta entidade é chamada de AC.

Um usuário pode solicitar um certificado diretamente a uma AC ou AR. Conforme Burnett e Paine (2002, p. 153), uma AR exerce as seguintes funções:

a) aceitar e verificar as informações de registro sobre novos registradores;
b) gerar chaves em favor de usuários finais;
c) aceitar e autorizar solicitações de backup e recuperação de chave;
d) aceitar e autorizar solicitações para revogação de certificado;
e) distribuir ou recuperar dispositivos de hardware como tokens, quando necessário.

Uma AR pode servir como uma entidade intermediária entre a AC e quem solicitou o certificado. Esta forma de obter um certificado comumente utilizada para usuários finais.

Para Silva (2004, p. 142), uma AC tem as seguintes obrigações:

a) manter rígida segurança para a chave privada da AC;
b) assegurar ampla distribuição de seu certificado;
c) emitir certificados;
d) revogar certificados;
e) emitir lista de certificados revogados;
f) publicar a lista de certificados revogados;
g) disponibilizar situação de certificado quando solicitada;
h) gerenciar de chaves criptográficas;
i) publicar suas regras operacionais;
j) fiscalizar o cumprimento da política pelos usuários.

Segundo Silva (2004, p. 141), as AC no Brasil, são estruturadas de maneira hierárquica com isso existe o conceito de autoridade raiz, ou seja, os certificados emitidos necessitam serem assinados pela AC raiz. É desta forma como o as AC estabelecidas do Brasil são estruturadas.

O certificado digital possui um tempo de vida, formado pela data de início e a data de validade. Um certificado necessita ter um prazo de validade devido à evolução dos dispositivos de processamento pela quantidade de vezes que foi usado.

Conforme a CertiSign (2006), os certificados digitais são utilizados por *sites* e aplicativos de rede para embaralhar os dados permutados entre dois computadores. A criptografia é uma ferramenta poderosa, mas, por si só, não constitui proteção suficiente para suas informações.

A criptografia não pode provar sua identidade ou a identidade da pessoa que está enviando dados criptografados. Pode-se tomar como exemplo um *site* de uma instituição financeira. O *site* pode até exigir que o usuário digite seu nome e sua senha, mas estes tipos de informações podem ser facilmente interceptados e não servem como prova da identidade. Sem garantias extras, alguém pode assumir a identidade e ter acesso a informações confidenciais.

Conforme Silva (2004, p. 144), existem vários certificados digitais no mercado mas o padrão que está dominando o mercado, e utilizado na ICP-Brasil, é o padrão X.509.

2.2.4 Certificados X.509

Conforme Silva (2004, p. 144), o padrão X.509 surgiu em 1988, como camada de autenticação recomendada para o padrão de diretório X.500 pelo ITU-T. O padrão X.500 pressupõe, como atributos de diretório, distribuição global e controle de acesso, isto significa identificação positiva e distinguida para autorização e possibilidade de sigilo nos canais de acesso. Esta primeira versão do padrão X.509 foi denominado de X.509 v1, prevalecendo até 1993, quando o padrão X.500 foi revisado. Após esta revisão o padrão X.509 v1 passou para segunda versão, denominado de X.509 v2, sendo o que difere da primeira versão para a segunda, é a inclusão de dois campos, o identificador único da AC e o identificador único do dono do certificado.

Em 1996 a especificação da terceira versão foi completada procurando com isso resolver a padronização do uso de chaves públicas, visto que a primeira e segunda versão não eram eficientes neste aspecto.

Segundo Silva (2004, p. 146), um certificado X.509 v3 é composto pelo pelas seguintes informações:

a) versão do certificado: identifica qual padrão é aplicado na versão do X.509 para

este certificado;

b) número serial: identifica a entidade que criou o certificado, podendo assim distinguir este de outros certificados emitidos por ele;

c) identificação do algoritmo de assinatura: identifica qual o algoritmo utilizado para gerar o certificado;

d) nome do emissor: identifica qual foi a AC que emitiu o certificado;

e) período de validade: identifica qual o período de validade do certificado;

f) nome do usuário: identifica a quem pertence a chave-pública;

g) chave-pública do usuário: identifica a chave-pública do usuário;

h) extensões: identifica a extensão e o tipo do dado;

i) um ou vários arquivos de extensão;

j) assinatura: é o código *hash* de todos os outros campos encriptados com a chave privada da AC.

Conforme Silva (2004, p. 147), as extensões permitem que uma AC inclua informações que não são normalmente oferecidas pelo conteúdo básico do certificado. Estas extensões possuem três componentes: identificadores, um sinalizador de criticidade e um valor. O identificador mostra o formato e a semântica do campo valor, o sinalizador de criticidade indica a importância da extensão, ou seja quando esse sinalizador estiver ligado indica que extensão é essencial para o uso do certificado, e caso um sinalizador de criticidade desconhecido for encontrado, o certificado não deve ser aceito.

Para Silva (2004, p. 149), uma extensão da versão 3 do padrão X.509 é composta pelos seguintes itens:

a) identificador único da chave da AC: essa extensão é utilizada para verificar a assinatura digital calculado no certificado;

b) identificador único associado com a chave pública: server para distinguir multiplas chaves que são distribuídas no mesmo certificado;

c) uso de chave: contém uma string de bits usada para identificar funções que podem ser suportados usando a chave pública;

d) uso de chave prolongado: contém a seqüência de um ou mais objetos que identificam o uso especifico da chave pública do certificado;

e) ponto de distribuição da LCR: indica a localização da lista de certificados revogados;

f) período de uso da chave privada: indica qual período de tempo que a chave privada associada com a chave pública desse certificado pode ser usada;

g) política de certificado: indica um ou mais objetos associados à política de emissão e uso do certificado;
h) nome alternativo: indica um nome alternativo para o dono do certificado como, e-mail;
i) nome alternativo para CA: indica nome alternativo para emissor do certificado como, e-mail;
j) restrição básica: indica se o certificado é para o usuário final ou para uma AC;
k) nome básico: extensão apenas presente em certificados da AC, indicando se são requeridos ou excluídos à cadeia de nomes da estrutura hierárquica;
l) política de restrição: extensão presente somente em certificados da AC, indicando a obrigatoriedade de políticas explicitas no caminho da certificação.

Segundo Silva (2004, p. 151), além destas extensões a RFC 3280 define ainda extensões privadas para serem usadas em domínios específicos sendo as seguintes: acesso de informação da autoridade e acesso de informação do sujeito.

3 DESENVOLVIMENTO DO TRABALHO

Neste capítulo são abordadas as atividades relativas ao projeto e o desenvolvimento do protótipo proposto.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O *middleware* desenvolvido obedece os seguintes requisitos funcionais e não funcionais:

a) o protótipo desenvolvido gera o par de chaves, onde uma é a chave pública e outra a chave privada (requisito funcional);
b) o protótipo emiti certificados digitais auto-assinados obedecendo ao padrão X.509 (requisito funcional);
c) o protótipo permiti a comunicação entre objetos distribuídos (requisito funcional);
d) o protótipo atende a requisição para emissão de certificados digitais para os objetos distribuídos, sendo que estes certificados obedecerão ao padrão X.509.Esta requisição ocorre apenas no início da comunicação (requisito funcional);
e) o protótipo revoga um certificado do objeto quanto este finalizar sua comunicação (requisito funcional);
f) o protótipo disponibiliza em tempo de execução uma lista com os certificados revogados, atualizando esta lista a cada certificado que for revogado (requisito funcional);
g) o protótipo desenvolvido foi implementado utilizando Java 5 (requisito não funcional);
h) o protótipo é compatível com sistema operacional Windows 2000 e XP (requisito não funcional).

3.2 ESPECIFICAÇÃO

Esta seção apresenta o diagrama de classe do *middleware*, o estudo de caso e o diagrama de seqüência do protótipo proposto. Para a confecção destes diagramas foi utilizada a ferramenta Enterprise Architect da Sparx Systems.

3.2.1 Diagrama de Classe

O diagrama de classe exibido na figura 6 mostra as classes responsáveis pela emissão do certificado digital para a raiz e o certificado digital para os objetos distribuídos.

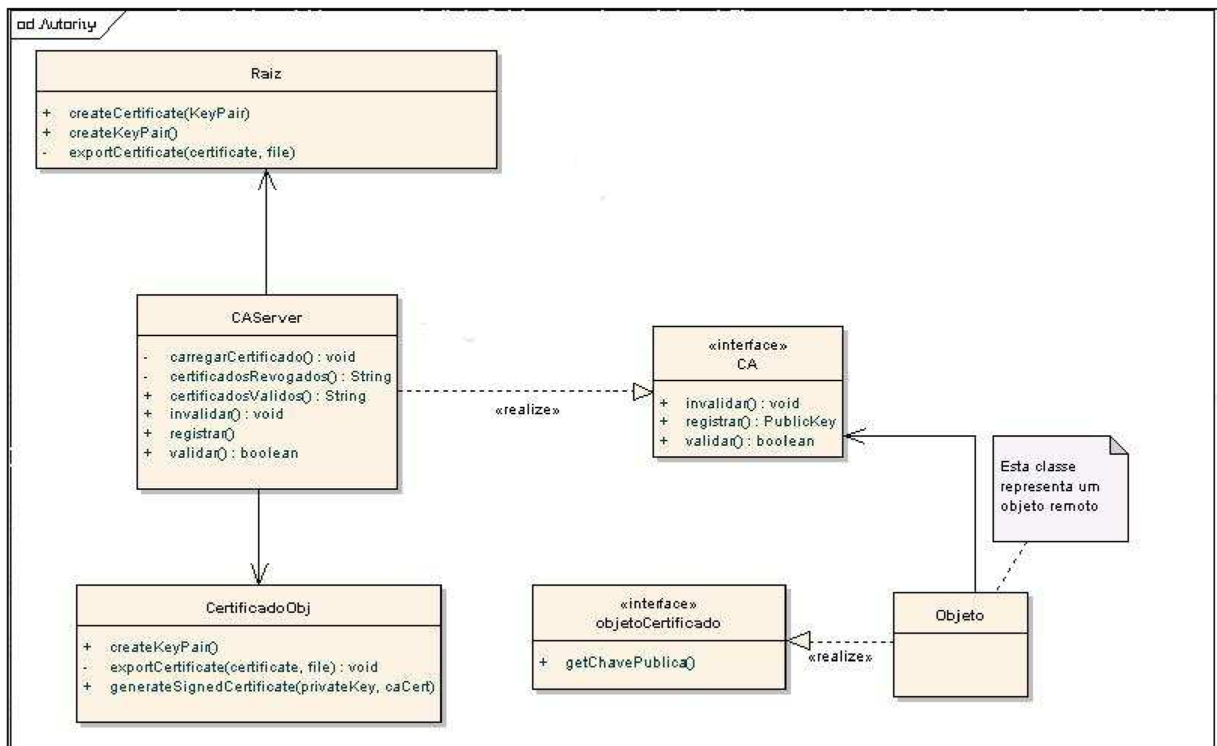


Figura 6 – Classes certificação digital

A figura 6 representa o relacionamento das classes do *middleware*, sendo que a descrição destas classes é apresentada nos tópicos abaixo.

3.2.1.1 Classe `Raiz`

A classe `Raiz` é a estrutura base para geração de certificado digital. Por meio desta

classe é gerado o certificado auto assinado, obedecendo com isso à hierarquia conforme descrito em 2.2.3.

A funcionalidade de emitir os certificados foi especificada no método *createCertificate(keyPair)*, onde os atributos do certificado são passados conforme necessidade da API *Bouncy Castle.* Através desta API pode-se emitir certificados auto-assinados ou certificados assinados por uma AC, e entre vários algoritmos de criptografia para assinar o certificado disponibilizado pela API, foi usado à junção do algoritmo MD5 com RSA.

Para emitir um certificado, necessita-se ter uma chave pública e privada, neste sentido foi especificado o método *createKeyPair()*, onde este método cria o par de chaves. O algoritmo utilizado para gerar o par de chaves foi o RSA utilizando tamanho da chave de 512 bits.

Mesmo não tendo necessidade de gravar o certificado em disco, optou-se por fazê-lo visando facilitar a visualização do mesmo. Neste caso o método *exportCertificate(certificate ,file)* foi especificado para gerar o arquivo do certificado digital e a extensão deste arquivo é *.crt*.

3.2.1.2 Classe CertificateObj

O objetivo desta classe é emitir certificados para os objetos distribuídos, sendo que estes certificados são assinados pela AC aqui representado pela classe raiz. Para emitir estes certificados foi especificado o método *generateSignedCertificate (*identificador, *privateKey, caCert)*, recebendo o identificador do objeto remoto, a chave privada da objeto remoto e o certificado da AC. O identificador é utilizado para dar nome ao certificado do objeto remoto, pois o identificador nada mais é que o *hascode* do objeto remoto, a *privateKey* é a chave privada e *caCert* o certificado da AC, sendo estes utilizados para assinar o certificado digital do objeto remoto.

Como na classe Raiz, para emitir o certificado denominado raiz, necessita-se criar um par de chaves, e para atender esta necessidade criou-se o método *createKeyPair(String alias)*. Conforme na classe raiz usou-se o algoritmo de criptografia RSA com tamanho de chave 512 bits.

Nesta classe conforme a classe raiz o certificado do objeto remoto é gravado em disco

através do médoto `exportCertificate(certificate ,file)`.

3.2.1.3 Classe CA

Devido à necessidade que os clientes possuem de manipular objetos residentes no servidor e considerando a impossibilidade de copiá-los, optou-se pela criação desta interface, para prover a ligação do cliente com os objetos residentes no servidor. Sua principal funcionalidade é a declaração das funções invocadas remotamente. Para isso, a interface deve ser estendida da classe *Remote*. Neste sentido optou-se por especificar os seguintes métodos, *registrar(o)*, *validar(o)*, *invalidar(o)*, possibilitando assim a comunicação com o servidor CAServer.

3.2.1.4 Classe `CAServer`

Através desta classe são emitidos os certificados para os objetos remotos e para a AC, disponibilizando os certificados na lista de certificados válidos ou na lista de inválidos e permite imprimir em vídeo os certificados revogados além de disponibilizar servidor para os objetos remotos obterem o certificado digital. Desta forma é inicialisado nesta classe o `RMIRegistry`, e registra o servidor através `Naming.rebind("//localhost/CA", obj)`.

Com a necessidade de criar primeiramente o certificado da AC, optou-se neste protótipo por implementar um método cuja finalidade é criar então o certificado auto assinado da AC. Neste sentido, o método `CAServer()`, atende a esta necessidade, permitindo assim que os certificados dos objetos remotos possam ser assinados pela raiz.

Para emitir os certificados digitais dos objetos remotos, desenvolveu-se um método capaz de atender esta necessidade, onde o método `registrar(o)` recebe o objeto remoto gera-se o par de chaves através do método `createKeypair(identificador)` da classe `certificateObj` e o certificado gerado e assinado pela AC pelo método `genereteSignedCertificate(identificador, keyPair.getPrivate(), raiz)`. Como para estabelecer comunicação entre os objetos remotos utilizando certificado digital, optou-se por criar uma lista de certificados válidos, ou seja, somente ira estabelecer comunicação se o certificado estiver nesta lista e para implementar esta lista foi desenvolvido o método `validar`

`(o)`. Para isso neste método pega a chave pública e o certificado do objeto, verifica se a data de vencimento do certificado não é maior que a data atual e o adiciona na lista de certificado válidos caso o certificado seja válido ou inválido caso seja inválido.

Conforme descrito em 2.2.3, faz-se necessário revogar um certificado digital, sendo que para isto foi desenvolvido o método `invalidar(o)`, onde este método recebe o objeto pega a chave pública, e remove seu certificado da lista de certificados válidos, e adiciona seu certificado na lista de certificados revogados. Neste protótipo optou-se por revogar o certificado do objeto remoto ao finalizar sua comunicação.

Através do método `certificadosRevogados()`, consegue-se listar o conteúdo da lista de certificados revogados, onde neste protótipo optou-se por listar apenas o identificados do certificado, ou seja a quem este certificado digital pertence.

3.2.1.5 Classe `ObjetoCertificado`

Para os objetos remotos obterem a sua chave pública, foi necessário especificar uma interface que atendê-se esta deficiência. Desta forma foi criado esta classe, servido de interface para os objetos, onde a chave pública do objeto é recuperada através do método `getChavePublica()`.

3.2.2 Especificação do estudo de caso

Para testar a funcionalidade deste protótipo, optou-se por especificar este estudo de caso, sendo esta especificação independente do *middleware*, e onde esta comunicação entre cliente e servidor representa um sistema de registro de notas acadêmicas. Num aplicativo de registro de notas no lado do cliente podem-se lançar as notas, fazer alterações ou excluí-las, já no lado do servidor juntamente com um banco de dados, estas notas podem ser armazenadas.

No protótipo especificado simula-se o lançamento das notas de acadêmicos, sendo assim foi especificado uma interface com dois métodos, o método `mensagem()` e `setNota(nota)` possibilitando com isso a comunicação a comunicação entre cliente e servidor. A figura 7 representa o diagrama de classe do estudo de caso.

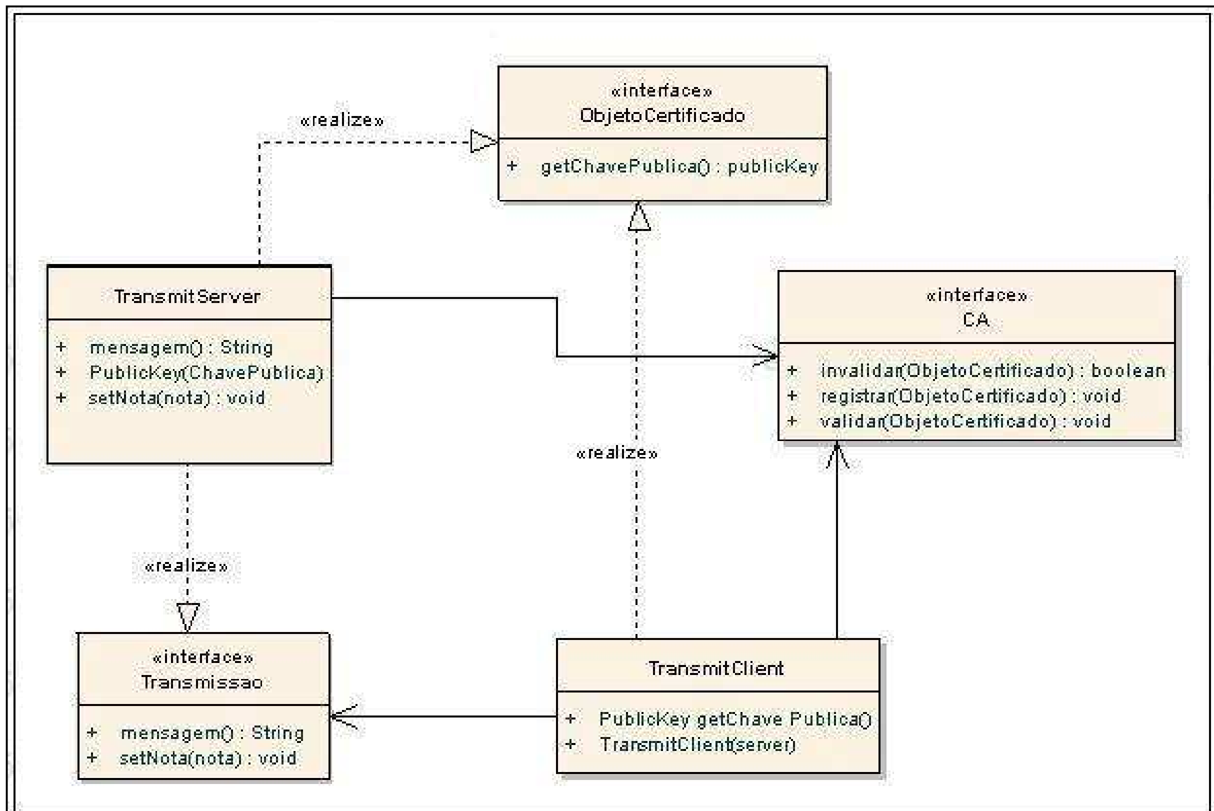


Figura 7 – Diagrama classe estudo de caso

Na figura 7 pode-se observar o relacionamento das classes da comunicação remota e o relacionamento com as interfaces do *middleware*, onde através destas interfaces consegue-se atender as solicitações dos objetos quanto sua certificação.

3.2.3 Diagrama de seqüência

A seguir é apresentado o diagrama de seqüência do protótipo especificado. Na figura 8 é apresentado o diagrama de seqüência da classe `CAServer` sendo instanciada.

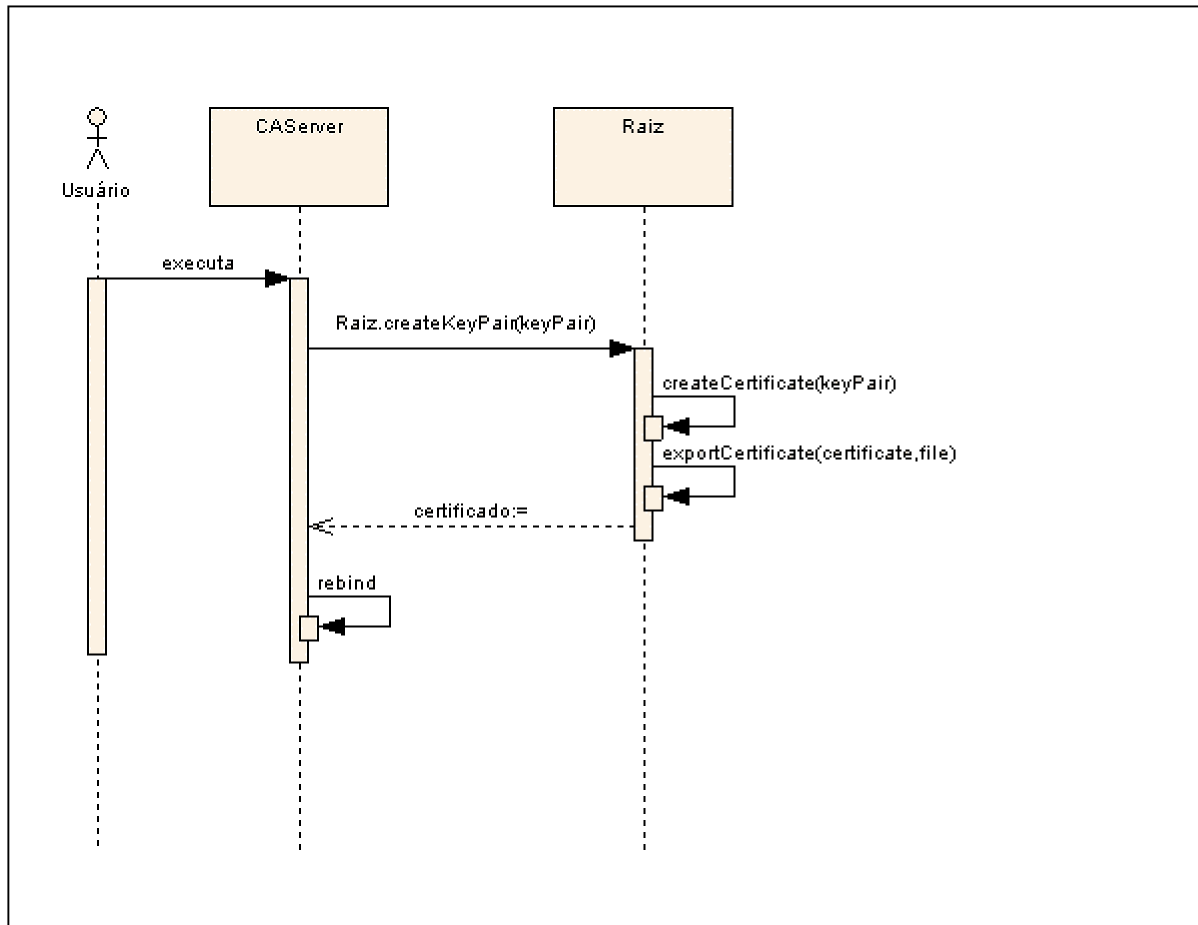


Figura 8 – Iniciando `CAServer`

Nesta figura pode-se observar que ao instanciar a classe *`CAServer`* é gerado o certificado digital da AC e é registrado o servidor para que os objetos remotos possam obter o certificado digital.

A figura 9 apresenta a classe *`TransmitServer`* sendo instanciada.

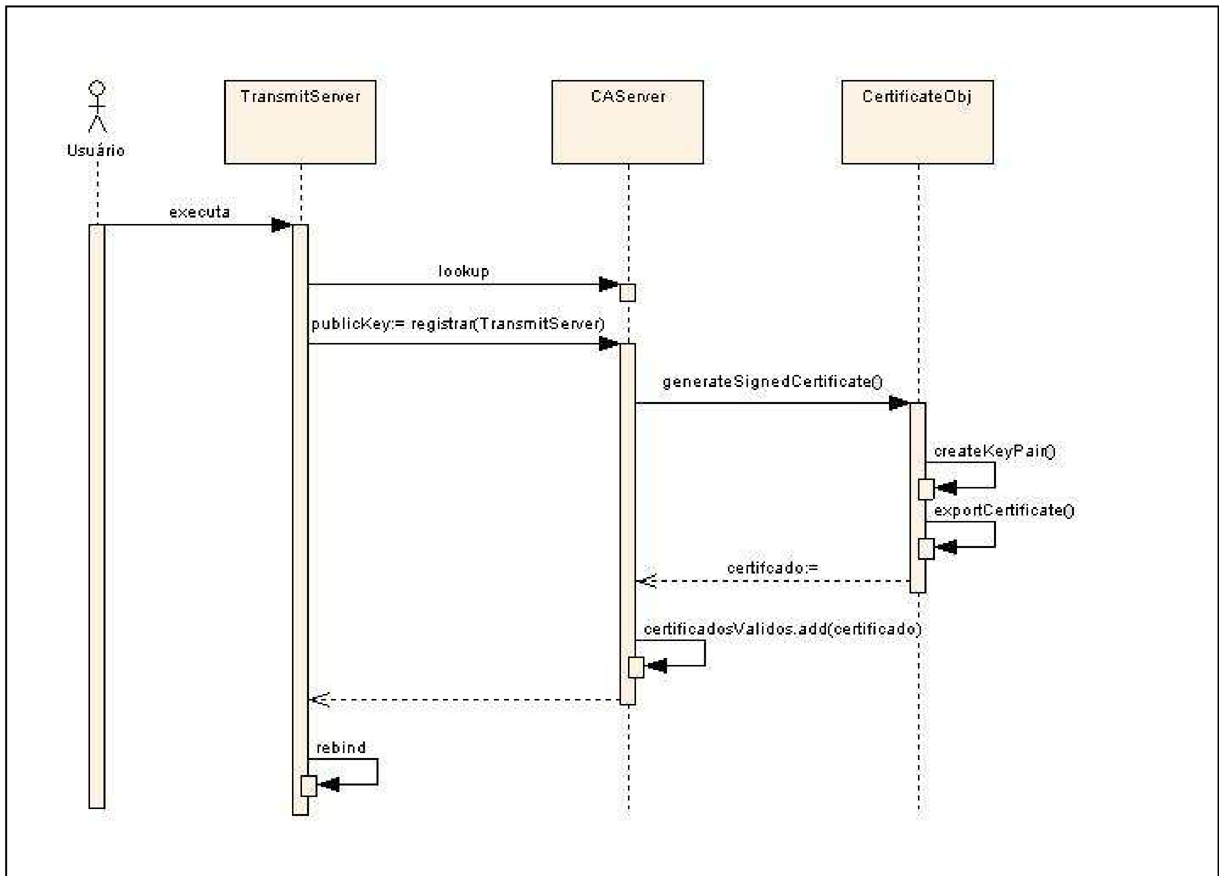


Figura 9 – Iniciado `TransmitServer`

Esta figura apresenta o servidor remoto pegando a referência do servidor do *middleware* para poder solicitar o seu certificado digital, o certificado ao ser gerado é adicionado a lista de certificados válidos, o servidor remoto se registra para poder estabelecer comunicação remota.

A classe `TransmitClient` sendo instanciada é apresentado na figura 10.

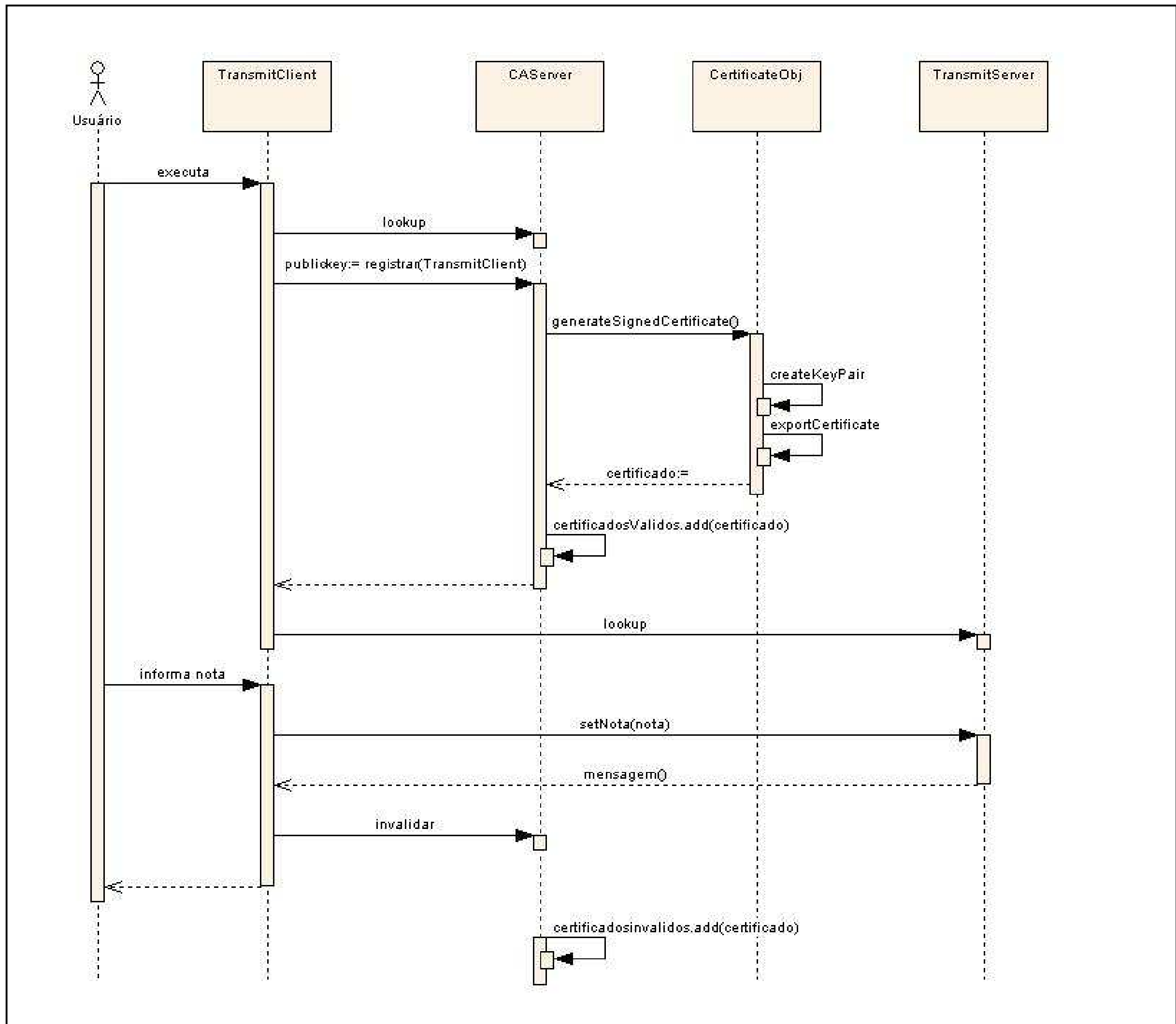


Figura 10 – Inicia `TransmitClient`

Esta figura apresenta o cliente pegando a referência do servidor para poder solicitar o seu certificado digital, o certificado ao ser gerado é adicionado a lista de certificados válidos, o cliente pega a referência do servido remoto para estabelece a comunicação, apresenta a troca de mensagens com o servidor remoto e o a solicitação de revogar o certificado digital.

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

O ambiente de desenvolvimento utilizado foi o Eclipse 3.2 com JVM 1.5 e API *Bouncy Castle*. Esta API foi escolhida porque disponibiliza toda infra-estrutura necessária para gerar certificado auto-assinado, assinar os certificados dos objetos distribuídos.

Para emitir o certificado digital da AC, foi desenvolvido o método `createCertificate`, conforme mostra o quadro 1.

```
...
public static X509Certificate createCertificate(KeyPair keyPair) throws Exception {
    // nome para AC Raiz
    String nome;
    nome="Autoridade Certificadora TCC";
    //provedor de seguranï¿½a do BouncyCastle
    Security.addProvider(new BouncyCastleProvider());
    //time from which certificate is valid
    Date startDate = new Date();
    //time after which certificate is not valid
    Date expiryDate = new Date();
    Calendar cal = GregorianCalendar.getInstance();
    cal.setTime(expiryDate);
    cal.add(Calendar.DAY_OF_MONTH, 1);
    expiryDate = cal.getTime();
    //serial number for certificate
    BigInteger serialNumber = new BigInteger("123456");
    // gera o certificado da raiz
    X509V1CertificateGenerator certGen = new X509V1CertificateGenerator();
    X500Principal dnName = new X500Principal("CN="+nome);
    certGen.setSerialNumber(serialNumber);
    certGen.setIssuerDN(dnName);
    certGen.setNotBefore(startDate);
    certGen.setNotAfter(expiryDate);
    certGen.setSubjectDN(dnName); // note: same as issuer
    // set chave publica
    certGen.setPublicKey(keyPair.getPublic());
    //especifica qual algoritmo de criptografia
    certGen.setSignatureAlgorithm("MD5withRSA");
    X509Certificate certRaiz = certGen.generate(keyPair.getPrivate(), "BC");
    //imprime certificado em console
    System.out.println(certRaiz);
    System.out.println("\n\n\n\n");
   //local de destino do arquivo
   exportCertificate(certRaiz, new File("C:\\"+nome+".crt"));
   return certRaiz;
}// fim do metodo createCertificate
...
```

Quadro 1 – Emissão certificado raiz

Neste método são atribuídos os valores como nome do certificado raiz data inicial data de vencimento sendo que os comandos para gerar o certificado são próprios da API. Para data de vencimento optou-se acrescentar um dia a data atual, e para isso foi utilizado a classe `GregorianCalendar`.

Como para emitir um certificado há a necessidade de criar um par de chaves, este par de chaves é criado através do método `createKeyPair()`, conforme mostra o código fonte do quadro 2.

```
public static KeyPair createKeyPair(){
  try{
       KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
       SecureRandom secRan = new SecureRandom();
       keyGen.initialize(512, secRan);
       return keyGen.generateKeyPair();
 }catch (NoSuchAlgorithmException e) {
       throw new RuntimeException(e);
 }
}
```

Quadro 2 – Emissão par de chaves

Neste método é gerado o par de chaves para a emissão do certificado digital, sendo que o algoritmo utilizado é o RSA e o tamanho da chave de 512 bits.

Neste instante temos o certificado em memória, não é necessário gravar o certificado em disco, mas neste protótipo optou-se por gravá-lo. O código fonte do método *exportCertificate(certificate, file)*,para gravar o certificado fisicamente é apresentado no quadro 3.

```
  private static void exportCertificate(X509Certificate certificate, File file)
           throws IOException, CertificateEncodingException,
           java.security.cert.CertificateEncodingException {
     FileOutputStream fos = new FileOutputStream(file);
     BufferedOutputStream bos = new BufferedOutputStream(fos);
     try {
           fos = new FileOutputStream(file);
           bos = new BufferedOutputStream(fos);
           bos.write(certificate.getEncoded());
           bos.flush();
     } finally {
                 if (fos != null)
                       fos.close();
                 if (bos != null)
                       bos.close();
               }
}
```

Quadro 3 – Grava certificado

O quadro 3 apresenta o método *exportCertificate*, onde através da classe *FileOutputStream* grava-se o arquivo em disco. Para garantir o correto armazenamento do arquivo, o conteúdo é armazenado em *buffer* até todas as informações pertencentes ao certificado forem processadas, obtendo assim o arquivo completo do certificado.

Concluída a etapa de emissão do certificado raiz, pode-se emitir certificados digitais para os objetos remotos, onde estes certificados são assinados pela AC. No quadro 4 apresenta-se o método da classe *CertificateObj*, para emissão dos certificados, os métodos para criar o par de chaves e gravar o certificado em disco são os mesmos da classe *Raiz*, implementados também na classe *CertificateObj*.

```
...
public static X509Certificate generateSignedCertificate(String identificador,
        PrivateKey privateKey, X509Certificate caCert) throws Exception {
    Date startDate = new Date(); // time from which certificate is valid
    Date expiryDate = new Date(); // time after which certificate is not valid
    Calendar cal = GregorianCalendar.getInstance();
    cal.setTime(expiryDate);
    cal.add(Calendar.DAY_OF_MONTH, 1);
    expiryDate = cal.getTime();
    BigInteger serialNumber = new BigInteger("654321"); // serial number for certificate
    KeyPair keyPair = createKeyPair(); // public/private key pair that we are creating certificate for
    X509V3CertificateGenerator certGen = new X509V3CertificateGenerator();
    X500Principal subjectName = new X500Principal("CN="+identificador);
    certGen.setSerialNumber(serialNumber);
    certGen.setIssuerDN(caCert.getSubjectX500Principal());
    certGen.setNotBefore(startDate);
    certGen.setNotAfter(expiryDate);
    certGen.setSubjectDN(subjectName);
    certGen.setPublicKey(keyPair.getPublic());
    certGen.setSignatureAlgorithm("MD5withRSA");
    X509Certificate certObj = certGen.generate(keyPair.getPrivate(), "BC");
    // imprime certificado no console
    System.out.print(certObj);
    //local de destino do arquivo
    exportCertificate(certObj, new File("C:\\"+identificador+".crt"));
    return certObj;
}// fim metodo generateSignedCertificate
...
```

Quadro 4 – Emissão certificado objeto

Este método recebe do identificador a chave privada e o certificado da raiz, emitindo assim o certificado para o objeto remoto, quanto a validade do certificado usou-se da mesma lógica aplicada para a emissão do certificado raiz.

Para estabelecer comunicação cliente servidor utilizando RMI, o `Registry` deve ser executado e um servidor registrado. No quadro 5 é demonstrado o código que implementa esta funcionalidade.

```
...
LocateRegistry.createRegistry(1099);

CAServer obj = new CAServer();
Naming.rebind("//localhost/CA", obj);
...
```

Quadro 5 – Registro Servidor *CAServer*

Após ter executado o `Registry` cria-se uma instância do `CAServer` e registra-se o servidor, podendo atender as requisições dos objetos distribuídos quanto a emissão de certificados digitais.

Na classe `CAServer` através do método registrar(o), é criado o certificado para o objeto remoto, este código é mostrado no quadro 6.

```
public PublicKey registrar(ObjetoCertificado o) {
  try {
      String identificador = "" + o.hashCode();
      KeyPair keyPair = CertificateObj.createKeyPair(identificador);
      PublicKey publicKey = keyPair.getPublic();
      Certificate certificate =
CertificateObj.generateSignedCertificate(identificador,keyPair.getPrivate(),raiz);
      certificadosValidos.put(publicKey, certificate);
      return publicKey;
      } catch (Exception e) {
            throw new RuntimeException(e);
        }
}
```

Quadro 6 – Método `Registrar`

Neste método é obtido o identificador do objeto remoto, gera-se o par de chaves para o objeto, cria o certificado digital sendo que para isso é passado o identificador, a chave privado do objeto e o certificado da raiz e adiciona o certificado gerado a lista de certificados validos.

Conforme necessidade de disponibilizar uma lista de certificados digitais validos, ou lista de certificados revogados, optou-se por desenvolver os métodos mostrados no quadro 7.

```
...
public boolean validar(ObjetoCertificado o) throws RemoteException {
      PublicKey publicKey = o.getChavePublica();
      X509Certificate certificate = (X509Certificate)certificadosValidos.get(publicKey);
      System.out.println("Certificados Validos: " + certificadosValidos());
      if (certificate == null) {
            return false;
      }
            Date datacertificado = certificate.getNotAfter();

      if (datacertificado.before(new Date())) {
            invalidar(o);
            return false;
      }

      return certificadosValidos.containsKey(publicKey);
}

public void invalidar(ObjetoCertificado o) throws RemoteException {
      Certificate certificate = certificadosValidos.remove(o.getChavePublica());
      certificadosRevogados.put(o.getChavePublica(), certificate);
      System.out.println("Certificados revogados: " + certificadosRevogador());

}
...

private Map<PublicKey, Certificate> certificadosValidos = new HashMap<PublicKey, Certificate>();

private Map<PublicKey, Certificate> certificadosRevogados = new HashMap<PublicKey, Certificate>();
...
```

Quadro 7 – Valida ou invalida certificado

Para verificar se o certificado do objeto é válido, recupera-se a chave pública do objeto e o seu certificado digital, atribuído para a variável *certificate*, podendo comparar desta forma se a data de vencimento do certificado ainda não fora atingida.

O método invalidar retira o certificado do objeto remoto da lista de certificados validos e o adiciona a lista de certificados revogados.

Para estabelecer a comunicação com o *CAServer*, necessita-se obter a referência do

servidor, sendo que o quadro 8 representa esta necessidade

```
...
CA ca = (CA) Naming.lookup("//localhost/CA");
this.publicKey = ca.registrar(this);
...
```

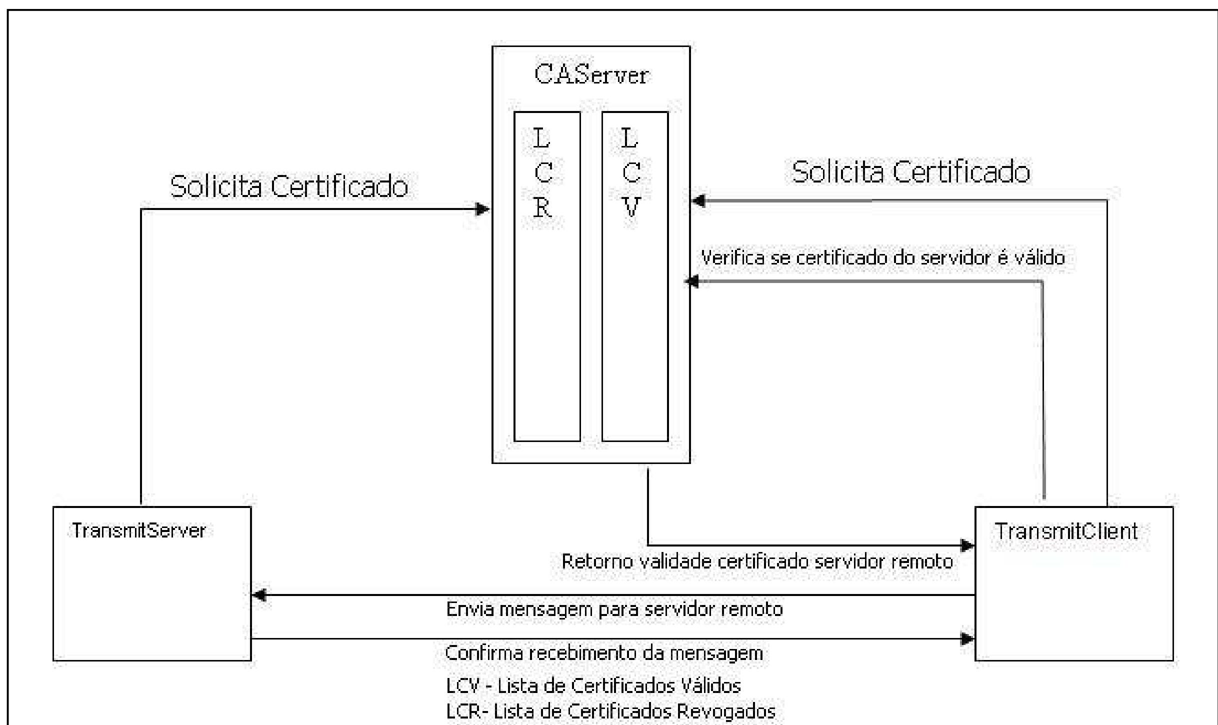
Quadro 8 – Referência servidor CA

Para obter a referência do servidor e sua interface, foi implementado o método `lookup`, podendo assim executar o método registrar implementado na interface *CA* obtendo desta forma o certificado digital para os objetos remotos.

3.3.2 Operacionalidade da implementação

Nesta seção são apresentadas as operacionalidades da implementação, demonstrando as etapas utilizadas para realizar a emissão de certificados digitais tanto para em nível de AC quanto aos objetos remotos.

O quadro 9 demonstra o funcionamento do protótipo.

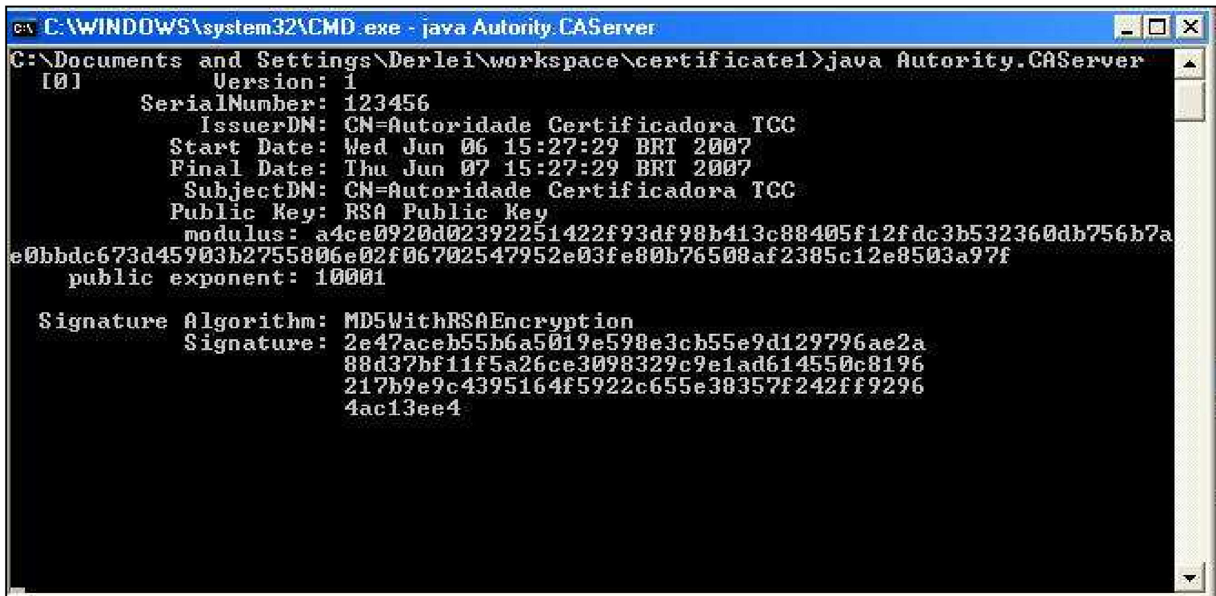


Quadro 9 – Funcionamento do protótipo

No quadro 9 pode-se observar o funcionamento do protótipo onde com *CAServer* atende as solicitações de certificação do servidor remoto e do cliente. Emitido o certificado

para o *TransmitServer* e *TransmitClient*, este certificado é adicionado a lista de certificados válidos, e quando *TransmitClient* troca mensagens com o *TransmitServer* é verificado se o certificado do *TransmitServer* está na lista de certificados válidos. Caso o certificado do *TransmitServer* for válido, *TransmitClient* manda mensagem para *TransmitServer*. Se o certificado do *TransmitServer* não for válido não ocorre troca de mensagens entre eles.

O primeiro passo é executar a classe `CAServer`, pois o método para criar o certificado da AC é instanciadao por esta classe. O quadro 10, demonstra o momento em que foi instanciada a classe `CAServer`.



Quadro 10 – Inicia `CAServer`

Com a execução da `CAServer`, emite-se o certificado da AC, ou seja o certificado-auto assinado e é registrado o servidor para que os objetos remotos possam obter um certificado.

No quadro 11 e 12 pode-se observar o certificado auto-assinado, onde este certificado representa a raiz, possibilitando assim a assinatura dos demais certificados.

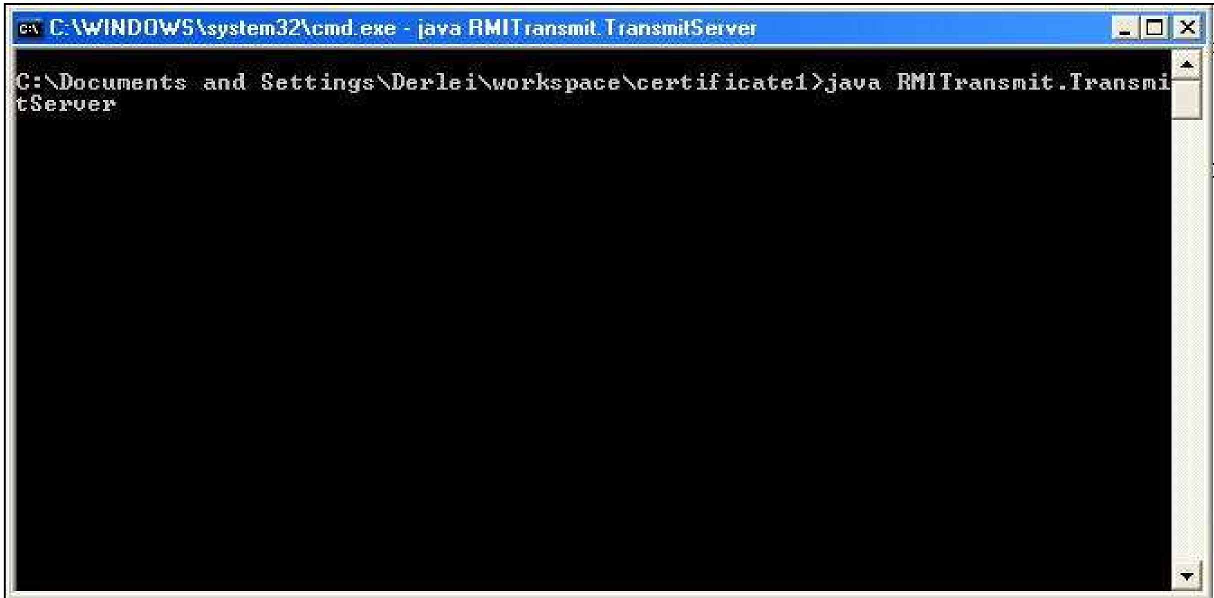
Quadro 11 – Certificado da AC geral

O quadro 10 apresenta o certificado digital da CA, de forma geral.

Quadro 12 – Certificado digital detalhes

No quadro 12 são apresentados os campos do certificado digital, como por exemplo, a versão, seu numero de série, o algoritmo de assinatura.

Após ter iniciado o `CAServer` e o certificado digital da AC ter sido gerado, pode-se iniciar o servidor `TramistServer`, onde nesta execução é gerado o certificado digital para o `TramistServer` e é registrado o servidor para comunicação com `TramistClient`.

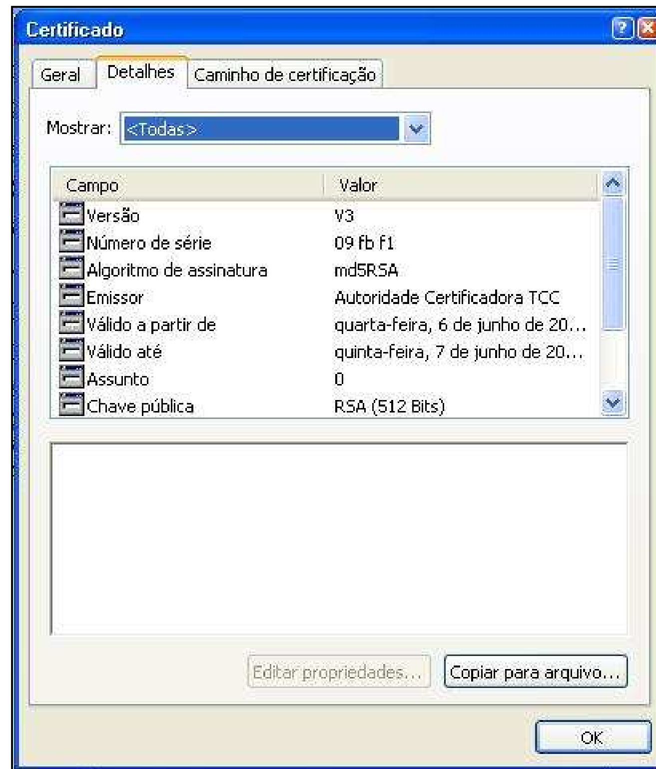


Quadro 13 – Inicia *TransmitServer*

O quadro 13 apresenta o servidor remoto rodando pronto para estabelecer comunicação com o cliente. No quadro 14 e 15, pode-se observar o certificado digital do `TramistServer`.

Quadro 14 – Certificado digital `TransmitServer` geral

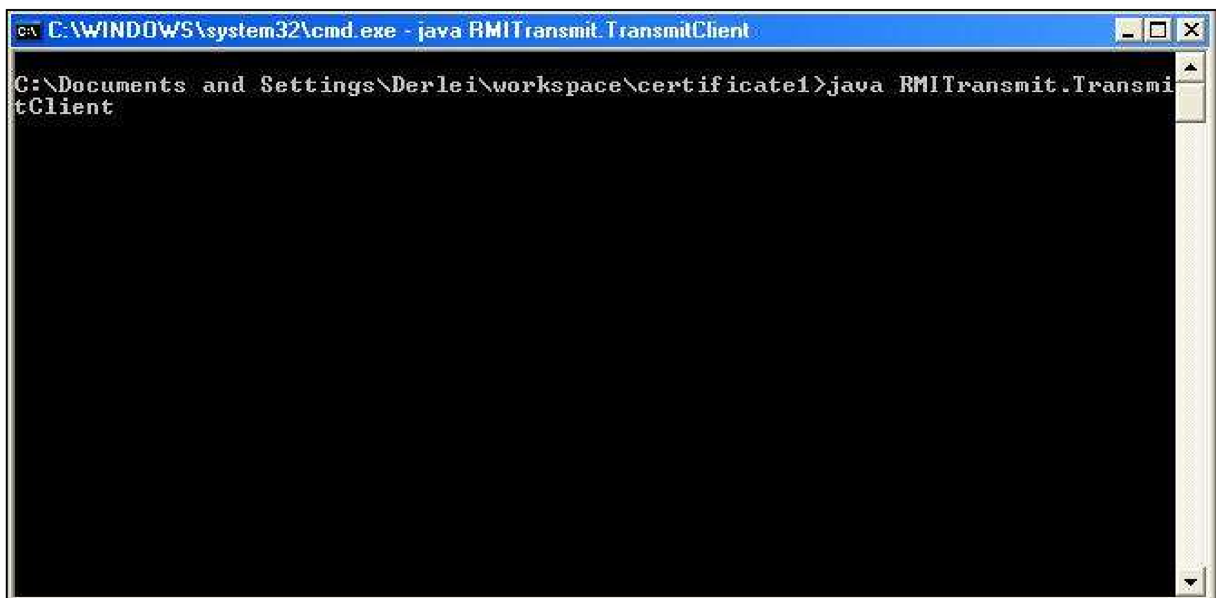
O quadro 14 apresenta o certificado digital do servidor remoto, onde pode-se observar quem assinou o certificado e a quem pertence.



Quadro 15 – Certificado digital `TramistServer` detalhes

No quadro 15 são apresentados os campos do certificado digital, como por exemplo, a versão, seu número de série e o algoritmo de assinatura.

Com os dois servidores rodando podemos iniciar o `TramistClient` o que é demonstrado no quadro 16.



Quadro 16 – Inicia `TramistClient`

O quadro 16 apresenta o `TransmitClient` rodando, e esperando que seja digitado algo para transmitir. O certificado gerado para o cliente, é demonstrado nos quadros 17 e 18.

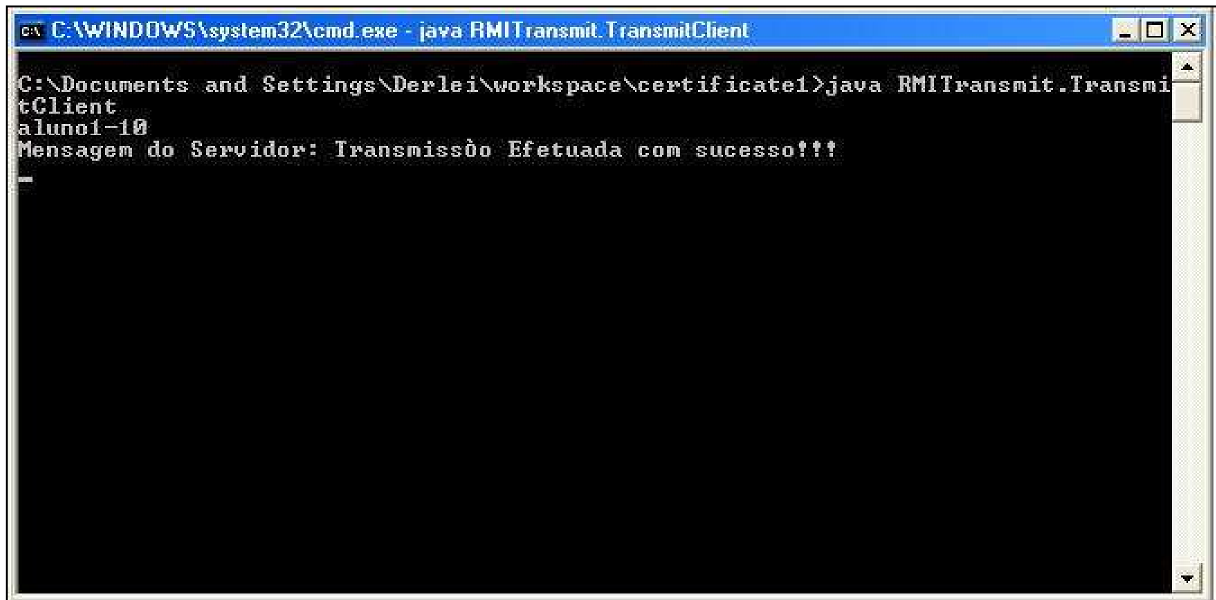
Quadro 17 – Certificado digital cliente geral

O quadro 17 apresenta o certificado digital do cliente, onde pode-se observar quem assinou o certificado e a quem pertence.

Quadro 18 – Certificado digital cliente detalhes

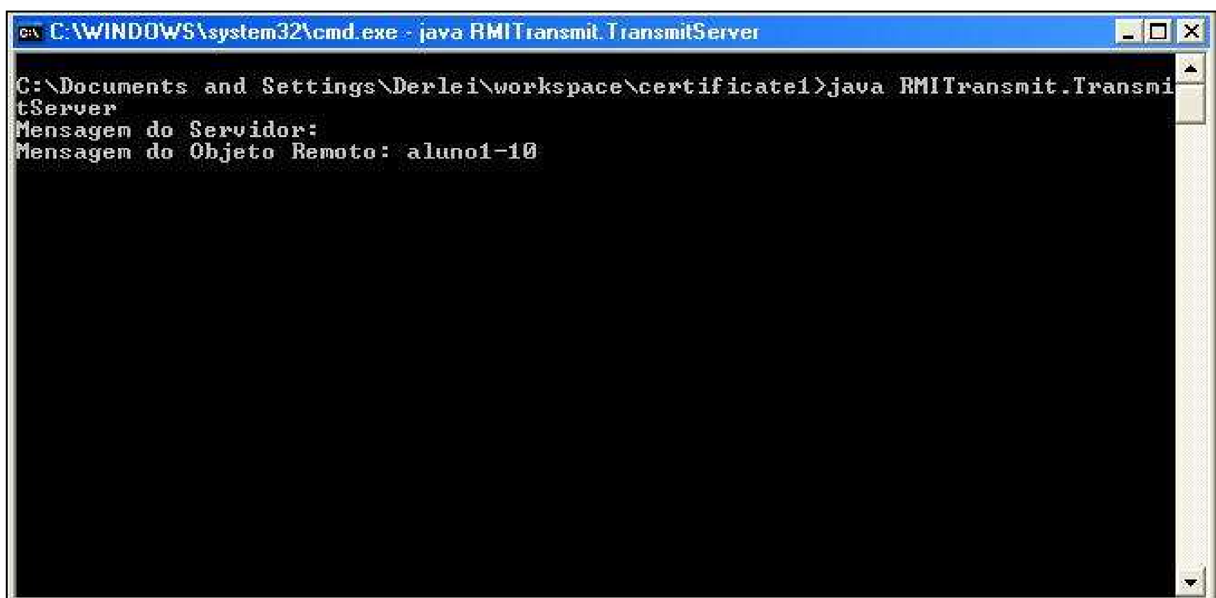
No quadro 18 são apresentados os campos do certificado digital, como por exemplo, a versão, seu numero de série, o algoritmo de assinatura.

Neste momento o `TramistServer` e o `TramistClient` tem seus certificados digitais, possibilitando assim a comunicação entre si. Esta troca de mensagens é demonstrada no quadro 19 e 20.



Quadro 19 – Troca mensagens cliente

O quadro 19 representa o cliente mandando uma mensagem para o servidor remoto.



Quadro 20 – Troca mensagens servidor

O quadro 20 representa o servidor remoto recebendo a mensagem do cliente.

Quando um cliente não tiver mais interesse em se comunicar basta mandar a mensagem fim, escrita em maiúsculo, onde desta forma encerra a comunicação e o seu certificado é revogado. Como um dos objetivos deste protótipo é disponibilizar uma lista de certificados revogados, esta lista contendo os certificados pode ser observada no quadro 21.

```
C:\WINDOWS\system32\cmd.exe - java Autority.CAServer
  [0]         Version: 3
        SerialNumber: 654321
            IssuerDN: CN=Autoridade Certificadora TCC
          Start Date: Wed Jun 06 16:11:27 BRT 2007
          Final Date: Thu Jun 07 16:11:27 BRT 2007
           SubjectDN: CN=22566565
          Public Key: RSA Public Key
           modulus: 957928b7ef42f33f48947b61f91d0537147e8236e123234df17e084f837
461f45cdafa55fd00310f0c1f87db55dc8b55df0739960997d4afb325eb7b53a7facd
    public exponent: 10001

  Signature Algorithm: MD5WithRSAEncryption
            Signature: 523bef7a666d60d1d344c49add5ec6977079ce97
                       7d4e9134a3b747e4842d8169abcc5a64afbf2d4f
                       6b86aaae5c518541be4ce260d78a9b65462c96a2
                       a00d3fbd

Certificados revogados: [CN=22566565]
```

Quadro 21 – Lista de certificados revogados

O quadro 21 apresenta a lista de certificados revogados, sendo que esta lista é apresentada no *CAServer*.

3.4 RESULTADOS E DISCUSSÃO

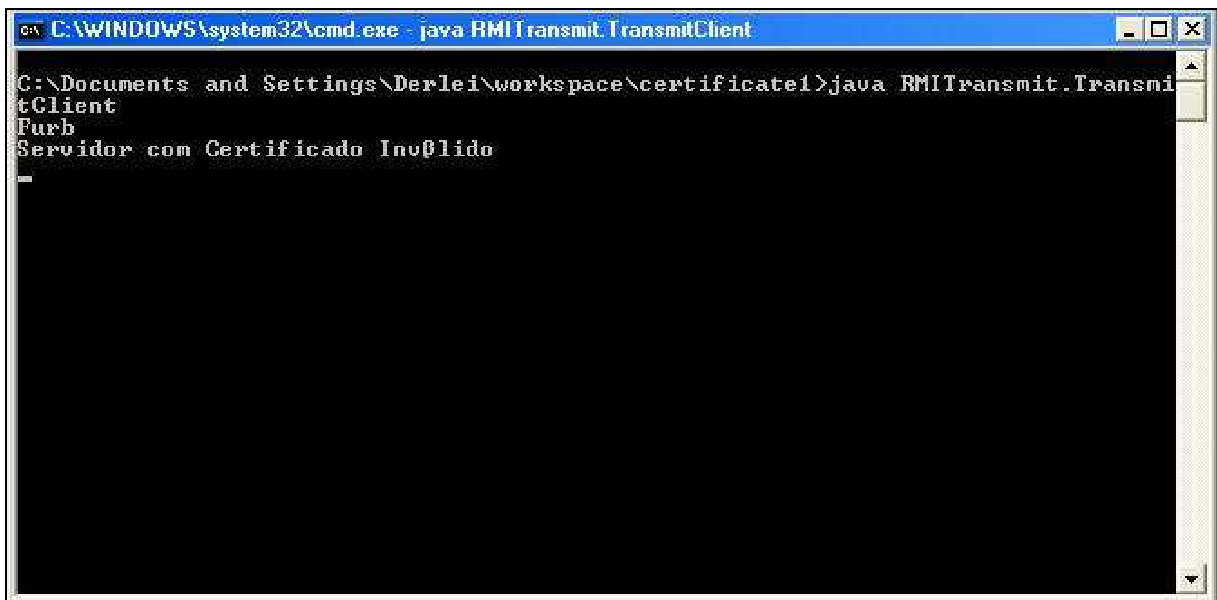
Neste trabalho foram citados quadro objetivos básicos para o funcionamento do *middleware* proposto, sendo assim, foi implementado uma estrutura para atender estes objetivos. Desta maneira, pode-se gerar o certificado digital auto-assinado, gerar o certificado digital para os objetos remotos, atender os objetos remotos quanto a solicitação de revogação de seu certificado, e mantendo uma lista em tempo de execução contendo os certificados revogados.

Em virtude do padrão de certificação X.509 ser amplamente conhecido, de fácil implementação e utilizado pela ICP-Brasil optou-se por seguir este padrão neste trabalho. Desta forma os certificados digitais seguem o padrão X.509 e modelo da ICP-Brasil, onde um certificado para ser válido necessita ser assinado por uma AC.

Neste protótipo a cada execução são gerados novos certificados digitais, deste o certificado da AC até os certificados dos objetos remotos, não persistindo a existência de certificados mesmo que estes certificados ainda estejam válidos.

Como neste protótipo não teve como objetivo utilizar o certificado digital de forma a criar um canal seguro entre cliente e servidor, sendo a única persistência feita na transmissão

remota é verificar se o certificado está válido. Um exemplo deste teste é demonstrado no quadro 22.



Quadro 22 – Servidor com certificado vencido

No quadro 22 apresenta o cliente querendo mandar uma mensagem mas não consegue pois o certificado do servidor esta vencido.

Para o desenvolvimento deste protótipo, foi pesquisado o trabalho de Ignaczak (2002), onde o mesmo descreve um novo modelo de infra-estrutura de chaves públicas para uso no Brasil, o trabalho de Parra (2002), que descreve metodologia para análise de segurança aplicada em uma infra-estrutura de chave pública e o trabalho de Hoppe (2005), recuperação distribuída de imagens por similaridade. Estes trabalhos tinham objetivos diferentes de forma que os mesmos foram utilizados como inspiração para o desenvolvimento deste trabalho.

4 CONCLUSÕES

Este trabalho apresentou alguns conceitos de criptografia, de sistemas distribuídos e o desenvolvimento do protótipo para emissão de certificados digitais.

As técnicas/ferramentas utilizadas foram adequadas ao desenvolvimento do trabalho, desde as tecnologias, como o Java RMI, o algoritmo de criptografia RSA e a API *Bouncy Castle*.

A utilização do algoritmo RSA foi importante, pois para emitir um certificado digital há a necessidade de ter um par de chaves. Através deste algoritmo também pode-se criptografar e descriptografar, assinar e validar uma assinatura.

Já a utilização da API *Bouncy Castle*, foi fundamental para a elaboração deste protótipo, pois esta API demanda infra-estrutura completa para emissão de certificados digitais auto-assinados e certificados digitais assinados por uma AC.

Como neste protótipo a comunicação entre objetos distribuídos foi implementado apenas para demonstrar a funcionalidade do *middleware*, sendo assim, não há preocupação em estabelecer um canal seguro de comunicação, desta forma a troca de mensagens entre os objetos remotos não é criptografada e a autenticidade de uma mensagem enviada entre eles também não é analisada, visto que estes tópicos não foram os objetivos deste protótipo.

De maneira geral, conclui-se que o trabalho contribui para o desenvolvimento de aplicações que utilizam certificado digitais, visto que o protótipo faz uso de técnicas importantes da segurança computacional.

4.1 EXTENSÕES

A complexidade e abrangência da certificação digital possibilitam que o presente trabalho seja estendido com diferentes objetivos. Durante o desenvolvimento do protótipo este item foi levado em consideração.

Primeiramente, pode-se criar a partir do protótipo desenvolvido, um *middleware* capaz de estabelecer um canal seguro entre objetos distribuídos, estabelecendo regras de distribuição do certificado digital entre os objetos, possibilitando com isso uma comunicação segura entre os objetos remotos.

Através deste protótipo pode-se criar uma AR, onde esta atende as requisições dos objetos remotos, identificando-os e validando-os, facilitando o trabalho da AC, ou seja a AR fica responsável pelo certificado do objeto remoto, deste a emissão até sua revogação e consequentemente mantendo a lista de certificados revogados. Desta forma uma AC não necessita emitir ou revogar vários certificado, cabendo a ela apenas o gerenciamento dos certificados das ARs.

Pode-se também com este protótipo desenvolver uma certificação cruzada onde um AC assina o certificado digital da outra AC, podendo ser uni ou bidirecional.

REFERÊNCIAS BIBLIOGRÁFICAS

ALBUQUERQUE, Fernando. **TCP/IP internet**: programação de sistemas distribuídos HTML, JavaScript e Java. Rio de Janeiro: Axecel Boocks, 2001.

BATTISTI, Júlio. **Tutorial TCP/IP**: parte 19. Santa Maria, [2003]. Disponível em: <http://www.juliobattisti.com.br/artigos/windows/tcpip_p19.asp/>. Acesso em: 08 jul. 2006.

BOUNCY CASTLE. **Java cryptography APIs**. Austrália, [2006]. Disponível em: <http://www.bouncycastle.org/java.html>. Acesso em: 09 maio. 2007

BURNETTI, Steve; PAINE, Stephen. **O guia oficial RSA**. Rio de Janeiro: Elsevier, 2002.

CASETTI Orestes; AKAMATU, Durval Makoto; KIRNER, Cláudio. **Paradigmas para construção de sistemas distribuídos**. Brasília, [1993]. Disponível em: <http://www.serpro.gov.br/publicacao/tematec/1993/ttec13/>. Acesso em: 17 set. 2006.

CERTISIGN. **Peticionamento eletrônico.** Rio de Janeiro, [2006]. Disponível em: <http://www.certisign.com.br/solucoes/trt4/trt4_faq.jsp#1>. Acesso em: 20 ago. 2006.

CENTRO DE ESTUDOS, RESPOSTA E TRATAMENTO DE INCIDENTES DE SEGURANÇA NO BRASIL. **Cartilha de segurança**. São Paulo, [2005]. Disponível em: <http://cartilha.cert.br/conceitos/sec8.html>. Acesso em: 21 out. 2006.

HOPPE, Aurélio Faustino. **Recuperação distribuída de imagens por similaridade**. 2005. 79 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

IGNACZAK, Luciano. **Um novo modelo de infra-estrutura de chaves públicas para uso no Brasil**: utilizando aplicativos com código fonte aberto. 2002. 125 f. Dissertação (Mestrado em Ciência da Computação) – Curso de Pós-graduação em Ciência da Computação, Universidade Federal de Santa Catarina, Florianópolis. Disponível em: <http://150.162.90.250/teses/PGCC0300.pdf>. Acesso em: 23 ago. 2006.

MONTEZ, Carlos. **Corba**. Florianópolis, [1998]. Disponível em: <http://www.das.ufsc.br/~montez/corba/corba.html>. Acesso em: 15 abr.2007.

PARRA, Gislaine. **Metodologia para análise de segurança aplicada em uma infra-estrutura de chave pública**. 2002. 97 f. Dissertação (Mestrado em Ciência da Computação) – Curso de Pós-graduação em Ciência da Computação, Universidade Federal de Santa Catarina, Florianópolis. Disponível em: <http://150.162.90.250/teses/PGCC0295.pdf >. Acesso em: 23 ago. 2006.

STALLINGS, William. **Cryptography and network security**. 3rd. ed. New Jersey: Pearson Education, 2003.

SILVA, Lino Sarlo da. **Public key infrastruture**. São Paulo: Novatec Editora, 2004.

SUM. **Java remote method invocation.** Santa Clara, [2007]. Disponível em: <http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp&sa=X&oi=translate&resnum=2&ct=result&prev=/search%3Fq%3D%2522java%2Brmi%2522%26hl%3Dpt-BR%26sa%3DG>. Acesso em : 15 abr. 2007.

TANENBAUM, Andrew S.; STEEN, Maarte van. **Distributed systems**: principles and paradigms. New Jersey: Prentice Hall, 2002.