Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Departamento de Engenharia Elétrica

Programa de Pós-Graduação em Engenharia Elétrica

Dissertação de Mestrado

Uma Arquitetura para Autenticação de Dispositivos Móveis através de uma Infra-Estrutura de Chave Pública

Jean Caminha

Orientadores

Angelo Perkusich, D.Sc.

Antonio Marcus Nogueira de Lima, Doutor

Campina Grande, Paraíba, Brasil

Novembro de 2006

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

C183u Caminha, Jean
2006 Uma arquitetura para autenticação de dispositivos móveis através de uma infra-estrutura de chave pública / Jean Caminha. Campina Grande: 2006.
101f.: il.

Referências.
Dissertação (Mestrado em Engenharia Elétrica) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.
Orientadores: Ângelo Perkusich e Antonio Marcus Nogueira de Lima.

1 Segurança da Informação 2 Autenticação 3 Infra-Estrutura de Chave Pública (PKI) 4 Protocolo de Iniciação de Sessões (SPI) 5 Dispositivos Móveis I Título

CDU 004.056.53

UMA ARQUITETURA PARA AUTENTICAÇÃO DE DISPOSITIVOS MÓVEIS ATRAVÉS DE UMA INFRA-ESTRUTURA DE CHAVE PÚBLICA

JEAN CAMINHA

Dissertação Aprovada em 22.12.2006

ANTONIO MARCUS NOGUEIRA LIMA, Dr., UFCG
Orientador

ANGELO PERKUSICH, D.Sc., UFCG
Orientador

Maria de Fatima Q. Vieira Turnell
MARIA DE FÁTIMA QUEIROZ VIEIRA TURNELL, Ph.D., UFCG
Componente da Banca

Benemar Alencar
BENEMAR ALENCAR DE SOUZA, D.Sc., UFCG
Componente da Banca

CAMPINA GRANDE – PB
Dezembro – 2006

Resumo

A identificação de objetos que participam de uma arquitetura de processamento de dados é uma preocupação relevante para a segurança das informações e fator decisivo para a utilização de serviços que utilizam dados sensíveis. Este trabalho propõe uma arquitetura para a autenticação de dispositivos móveis utilizando a Infra-estrutura de Chave Pública (PKI) e o Protocolo de Iniciação de Sessões (SIP) de modo a minimizar as limitações de armazenamento e processamento destes aparelhos.

Palavras-chave: Autenticação, Segurança da Informação, Infra-estrutura de Chave Pública (PKI), Protocolo de Iniciação de Sessões (SIP), Dispositivos Móveis.

Abstract

The identification of objects that participates of architecture of data processing is a concern for information security and decisive factor for services that need manipulate sensible data. This work considers architecture for mobile devices authentication through in a Public Key Infrastructure and the Session Initiation Protocol (SIP) as a way to minimize storage and processing limitations of those devices.

Key-words: Authentication, Information Security, Public Key Infrastructure (PKI), Session Initiation Protocol (SIP), Mobile Devices.

Dedicatória

Ao meu pai, Francisco Valdemir Caminha, exemplo.

Agradecimentos

Agradeço à Deus, por sua presença, proteção e apoio na realização deste trabalho e em todos momentos de minha vida. E minha mãe, Ritinha de Cássia Caminha (*in memorian*), que sempre intercede por mim junto a Ele.

Aos meus orientadores, Angelo Perkusich, D.Sc. e Antonio Marcus Nogueira de Lima, Doutor, referências, pela atenção e profissionalismo.

À Silane Aparecida Ferreira da Silva, companheira e amiga, por todo apoio, orientação, cobrança e confiança para que eu concluísse este trabalho.

À minha Família, mesmo privados de minha presença, sempre confiaram e apoiaram meus sonhos.

Ao Raul Herbster, incansável, por desvendar os segredos do Java neste projeto.

Aos colegas Danilo Santos e Olímpio Cipriano, pelas considerações quanto ao trabalho e ensinamentos de SIP.

À Dra. Isa Assef dos Santos e o Dr. Evandro Vieiralves, por acreditarem em mim e na realização deste trabalho.

Ao meu chefe, Miguel Grimm do Nascimento, por não medir esforços para viabilizar a conclusão deste trabalho, dicas valiosas e por acreditar na formação dos seus colaboradores.

Aos amigos Rodrigo Marques Jr. e Rhandsaissem Leal, que trabalharam por três, dando-me a tranqüilidade necessária para este projeto.

Aos amigos Christopher Xavier e Patrickson Santos, pela amizade, gentileza e apoio, recebendo-me em Campina Grande.

Ao amigo Rodrigo Choji de Freitas, pelo seu exemplo de disciplina e por ser companheiro nessa empreitada.

Ao Professor Antonio Cauper Filho, amigo, conselheiro e incentivador nesse processo.

Aos colegas do Laboratório de Computação Pervasiva (PERCOMP), Saulo Luiz, Kyller Gorgônio e Walter Silva, pela companhia e ajuda durante a realização deste trabalho.

À Fundação Centro de Análise, Pesquisa e Inovação Tecnológica - FUCAPI, por ser esta empresa diferenciada que acredita na formação das pessoas como combustível de seu desenvolvimento, na qual tenho a honra de trabalhar.

À Universidade Federal de Campina Grande – UFCG, por manter o alto nível do curso, apoio, organização e pelo empreendedorismo de levar um programa dessa qualidade ao Amazonas.

À Universidade do Estado do Amazonas – UEA e Superintendência da Zona Franca de Manaus – SUFRAMA, pelo financiamento e visão.

A todos os colegas do mestrado e também àqueles que colaboram e não foram citados, meu muito obrigado.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de siglas

3G	*Third generation of developments in mobile communications technology*
AC / CA	*Certification Authority / Autoridade Certificadora*
CDMA	*Code Division Multiple Access*
EDGE	*Enhanced Data GSM Environment*
LDAP	*Lightweight Directory Access Protocol*
MANET	*Mobile ad hoc network*
PDA	*Personal Digital Assistant*
PGP	*Pretty Good Privacy*
PKC	*Public Key Cryptography*
PKI	*Public Key Infrastructure*
RF-ID	*Radio Frequency Identification Device*
SIP	*Session Initiation Protocol*
SSL / TLS	*Secure Socket Layer / Transport Layer Security*
TDMA	*Time Division Multiple Access*
WCDMA	*Wideband Code-Division Multiple Access*
WI-MAX	*World Interoperability for Microwave Access*

Capítulo 1 - Introdução

1.1. Contextualização do trabalho

Os dispositivos móveis proporcionam um aumento das possibilidades de aplicações dos sistemas computacionais. Ao mesmo tempo em que são criados *hardwares* com alto poder de processamento, surgem equipamentos portáteis de alta performance com diferentes funções de mobilidade. Estes dispositivos, devido sua portabilidade, seus vários tipos de sensores e interfaces de comunicação, criam uma experiência computacional singular. Estes dispositivos estão ganhando popularidade e sofrendo importantes melhorias em seus *hardwares*, elevando seu poder computacional a cada ano [KOCH 03].

O desenvolvimento dos dispositivos móveis proporciona os usuários a interagirem com o processamento da informação independente dos locais onde se encontram e criam novas possibilidades de aplicações. A utilização da computação independente do lugar ou circunstâncias é o objeto de estudo da computação pervasiva. Dentro das aplicações possíveis, podemos enumerar sua utilização para o comércio eletrônico e interação com outros sistemas [KOUR 03], dispositivos para o cuidado da

saúde [DAVI 02], automação, coleta de dados e mensageria (mensagens instantâneas e multimídia) [EPP 03].

As pesquisas de novas tecnologias para os dispositivos móveis referem-se principalmente em deixar estes aparelhos mais autônomos e conectados possível. A capacidade de processamento e armazenamento destes aparelhos também é relevante, dependendo da função que irão executar. Dentre as tecnologias de conexão atualmente utilizadas, destacamos os padrões WCDMA, EDGE, TDMA, CDMA, 3G, 802.11, WI-MAX, Bluetooth e infravermelho [STAN 02]. Os dispositivos móveis também se valem de recursos como câmeras, microfones e leitores de etiquetas RF-ID para captarem informações do meio externo.

1.1.1. Problemática na utilização de PKI em dispositivos móveis

A capacidade de mobilidade e diferentes tipos de utilização destes dispositivos proporcionam uma preocupação aos usuários e desenvolvedores de aplicações com relação à segurança de informações. A segurança de informações baseia-se em proteger os dados trafegados e armazenados em três dimensões principais: Confidencialidade, integridade e disponibilidade. Além destes atributos, alguns autores também destacam o não-repúdio e a autenticidade. [HARR 05]. Dentro desse contexto, a confidencialidade oferece suporte à prevenção de revelação não autorizada de informações, além de manterem dados e recursos ocultos a usuários sem privilégio de acesso. Já a integridade previne a modificação não autorizada de informações. Por outro lado, a disponibilidade prover suporte a um acesso confiável e prontamente disponível a informações. Isto implica em dados e sistemas prontamente disponíveis e confiáveis. Adicionalmente, o

não repúdio e autenticidade compreendem o que poderia ser denominado de responsabilidade final e, dessa forma, busca-se fazer a verificação da identidade e autenticidade de uma pessoa ou agente externo de um sistema a fim de assegurar a integridade de origem.

No âmbito da computação móvel, a segurança da informação é uma preocupação constante, visto que o processamento ocorre independente do local ou circunstâncias e quase sempre utiliza recursos de conexões sem fio, o que eleva os riscos a exposição de dados e outros tipos de ataques, como negação de serviços e roubo de identidades [KOCH 03].

Outra preocupação relevante é a confiança nos usuários, dispositivos, serviços e aplicativos utilizados [CAHI 03]. É necessário haver confiança entres as partes envolvidas no sistema e também não repúdio das ações executadas. Por exemplo, podemos citar o banco eletrônico, onde tanto a instituição financeira precisa ter certeza que um determinado usuário é seu cliente quanto o cliente confia que está acessando os serviços de seu banco [DATT 03].

Dentre as ferramentas mais utilizadas pelo mercado para dirimir o problema da identificação e autenticação dos usuários e serviços podemos destacar a utilização de uma Infra-estrutura de Chaves Públicas (PKI) e cadeias de confiança do tipo PGP, que utilizam uma abordagem estatística para elevar o nível de confiança. A Infra-estrutura de Chaves Pública é a mais utilizada e aceita por vários serviços e segmentos do mercado. Todos os navegadores de internet mais utilizados possuem suporte a esta tecnologia. Várias empresas assinam os certificados emitidos, dando fé e garantindo a identidade dos solicitantes, atuando como uma terceira parte de confiança [CHOU 02, DATT 03].

A PKI baseia-se em um modelo de chaves assimétricas onde a confiança surge na comparação matemática de duas chaves complementares. Para tanto, é necessário que se possua um repositório contendo as chaves das diversas autoridades certificadoras e/ou chaves públicas de outros usuários. Este repositório de chaves possui uma vantagem de provocar resiliência, o que facilita sua consulta de modo off-line, sem a necessidade de utilização de servidores. Entretanto, por possuir este repositório, sua utilização nos dispositivos móveis é limitada em virtude da pequena capacidade de armazenamento que possuem estes aparelhos [CHOU 02].

Estes fatos evidenciam um problema na questão da utilização de certificados digitais de uma infra-estrutura de chaves pública (PKI) em dispositivos móveis, o que exclui estes aparelhos de padrões de mercados já estabilizados, corroborando para a dificuldade de criação de aplicativos e sistemas, principalmente aqueles que necessitam de autenticação e não repúdio.

1.2. Relevância do tema e escopo da pesquisa

Neste trabalho, será apresentada uma solução para autenticação de um usuário de um dispositivo móvel, tendo sua identidade validada por uma autoridade certificadora em uma infra-estrutura de chave pública (PKI). A autenticação de objetos é fundamental para segurança das informações, visto os problemas relacionados ao roubo de identidade e repúdio das ações realizadas por um usuário dos sistemas.

As empresas prestadoras de serviços digitais, como bancos, sítios de comércio eletrônico, dentre outros, necessitam saber quem são seus usuários e imputar responsabilidades por suas ações. Por outro lado, usuários precisam saber se estão

realmente acessando o serviço desejado, se uma aplicação instalada foi produzida por quem realmente publicou e se um outro usuário que deseja comunicar com ele é que diz que seja.

A relevância deste trabalho está na proposição de uma solução para utilização da infra-estrutura de chaves pública (PKI), uma arquitetura já aceita pelo mercado, por dispositivos móveis, aumentando o horizonte para criação de sistemas e aplicações no campo da computação pervasiva.

1.3. Objetivo do trabalho

O objetivo deste trabalho é propor uma solução para a utilização da infra-estrutura de chaves públicas (PKI) em dispositivos móveis. Com este serviço, espera-se prover o uso destes aparelhos em uma arquitetura de autenticação e não-repúdio amplamente utilizada e aceita pelo mercado. Por conta disso, o reuso de chaves públicas e lista de revogação também será possível, trazendo mais confiança e padronização ao ambiente.

Como objetivos específicos, este trabalho também irá:

- Levantar soluções existentes para a utilização da infra-estrutura de chaves pública em dispositivos móveis;
- Propor uma arquitetura para a utilização da PKI em dispositivos móveis;
- Implementar um estudo de caso para a utilização da infra-estrutura de chaves pública em dispositivos móveis e analisar os resultados;

Este trabalho apenas tratará da autenticação de objetos e não repúdio. Outros aspectos da segurança das informações como confidencialidade, integridade e disponibilidade não serão abordados.

1.4. Estrutura da dissertação

Esta dissertação está organizada da seguinte forma:

No Capítulo 2 é abordada uma revisão bibliográfica da PKI, fundamentando teoricamente o trabalho e avaliando outras soluções existentes para a utilização dos certificados digitais de uma infra-estrutura de chaves pública em dispositivos móveis.

No Capítulo 3 é proposta e definida uma arquitetura para a o serviço de certificados digitais e discutido as tecnologias utilizadas.

No Capítulo 4 é apresentado um estudo de caso, implementando a arquitetura proposta.

Finalmente, no Capítulo 5, uma discussão acerca das conclusões e resultados obtidos, bem como são apresentadas sugestões para trabalhos futuros.

Capítulo 2 – Conhecimentos de Suporte

O objetivo deste capítulo é apresentar uma introdução aos temas investigados e estudados ao longo deste trabalho. Foi realizada uma revisão bibliográfica e identificação de trabalhos correlatos ao assunto desenvolvido nesta dissertação.

2.1. Infra-estrutura de Chave Pública (PKI)

A infra-estrutura de chave pública (*Public Key Infrastructure, PKI*) é a tecnologia escolhida pelo mercado para a utilização do método de criptografia de chaves públicas (*Public Key Criptography, PKC*) que garante a segurança das comunicações digitais através da verificação de identidades e controle de encriptação em transações comerciais [NOAK 01]. O volume do comércio na internet requer um método robusto e eficiente para verificar a identidade da pessoa com quem uma está fazendo o negócio e proteger informações sensíveis. PKI é a infra-estrutura que fornece segurança do PKC e serviços de assinaturas digitais. É o arcabouço onde ocorre processo de controle de chaves, criação e distribuição de certificados digitais.

O desafio de verificar a identidade enquanto realiza transações com pessoas desconhecidas não é nenhum fato novo. Bem antes da internet, tais problemas também precisavam ser resolvidos. Por exemplo, viajar de um país para o outro requer um passaporte, bem como as transações em papel frequentemente devem ser chanceladas por um cartório. Em ambos os casos, entretanto, o original é aceito porque o documento foi atestado por uma terceira parte confiável. Para o passaporte, o país emissor requer diversas informações do solicitante para atestar sua identidade, antes de expedir o documento. Os outros países confiam no processo de emissão do país de origem e, por conseguinte, confiam na identidade do cidadão. Dentro do contexto de uma infra-estrutura de chave pública (PKI), este mesmo nível da confiança é possível. Os certificados digitais emitidos por uma autoridade certificadora (*Certification Authority, CA*) provêem confiança nas relações entre consumidores, fornecedores e parceiros de negócio [NOAK 01].

2.1.1. Funcionamento da Infra-estrutura de chave pública (PKI)

O conceito da infra-estrutura de chaves públicas (PKI) vem sendo explorado erroneamente pelo mercado. Segundo Noaks-Fry [NOAK 01], muitos fornecedores vendem a PKI como apenas um produto. A PKI, assim os sistema telefônicos e de energia elétrica, é um sistema feito de serviços, produtos e processos para alcançar os objetivos de segurança. Infra-estruturas estão em toda parte, ainda que transparentes, transportando informação, produtos, serviços e pessoas. Muitas pessoas somente percebem a existência de uma infra-estrutura apenas quando esta falha.

Quando um cliente compra um produto pela internet, vários elementos de uma PKI são colocados em ação. Um certificado é baixado do servidor para o cliente, o computador do cliente verifica a autenticidade do certificado e envia uma chave de volta ao servidor, para encriptar a sessão. Todas estas ações acontecem sem que o cliente perceba. E é esta transparência que torna eficiente uma infra-estrutura.

Outro tipo de infra-estrutura utilizada para validação de identidades é a de chaves simétricas compartilhadas. Chaves secretas (ou simétricas), usando chaves compartilhadas, vêm sendo usadas há algum tempo por bancos e redes comunicação privadas para transações ponto-a-ponto, como a comunicação com os caixas automáticos. Nesta arquitetura, cada ponto envolvido na comunicação possui uma chave para desencriptar as informações recebidas do outro ponto. Também é necessário possuir uma chave reconhecida pelo destinatário para realizar a transmissão de dados.

Caso o número de pontos de comunicação seja muito grande, a solução de compartilhamento de chaves torna-se impraticável devido ao número de chaves emitidas e gerenciadas. Noaks-Fry [NOAK 01] exemplifica que se 100 empresas desejassem se comunicar usando chaves secretas, 4.950 chaves deveriam ser criadas, gerenciadas e salvaguardadas (99 para a primeira empresa, 98 para a segunda, etc.). Por outro lado, em um ambiente de chaves pública, seriam necessários apenas 100 certificados.

No coração da autenticação provida pela PKI está o conceito de *public key cryptography* (PKC), ou criptografia de chave pública, como uma alternativa de gerenciar de forma eficiente as várias chaves distribuídas pelos vários sistemas e pontos. Em 1976, Whitfiel Diffie, Martin Hellman e Ralph Merkle apresentaram a criptografia de chave (assimétrica) pública (PKC) como proposta aos desafios de gerenciamento dos tradicionais esquemas de criptografia baseados em chaves simétricas. [CHOU 02, HARR 05] Na PKC, para cada pessoa, entidade ou objeto, é emitido um par de chaves:

- Uma **chave pública**, que pode ser acessada por todos (como um número de telefone), e;
- Uma **chave privada**, que é conhecida apenas pelo individuo para quem foi emitida a chave. Esta chave nunca é revelada ou transmitida.

O que uma chave do par encripta, a outra desencripta. As chaves estão matematicamente relacionadas de maneira que é virtualmente impossível adivinhar uma chave a partir de outra. [NOAK 01] O usuário pode distribuir a chave pública indiscriminadamente e deve proteger sua chave privada.

As principais aplicações das chaves assimétricas são:

- **Confidencialidade.** A pode enviar uma mensagem privada para B encriptando a mensagem com a chave pública de B, porque B possui a chave privada para desencriptá-la.
- **Autenticidade**. A pode enviar para B uma mensagem encriptada com a chave privada de A, que será desencriptada por B com a chave pública de A. A identidade do remetente e verificada porque somente A tem acesso à chave privada. Este é o princípio da assinatura digital.

A Figura 2.1 descreve o processo de assinatura digital, através do método de compartilhamento de chaves assimétricas. Este método é muito utilizado em sistemas privados de criptografia.

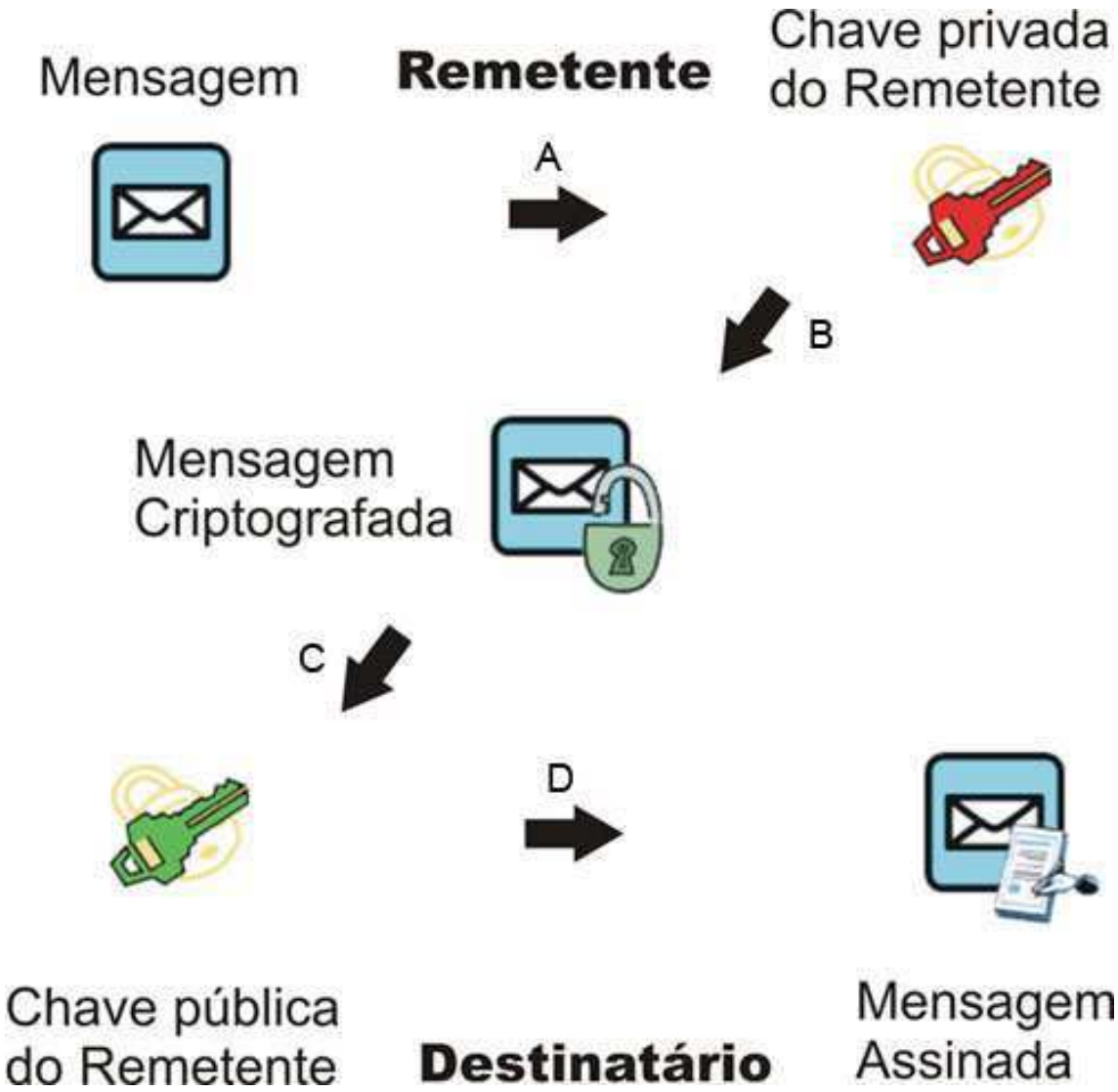

Figura 2.1 - Processo de assinatura eletrônica.

Para verificar uma assinatura, a mensagem a ser enviada é criptografada com a chave privada do remetente que só pode ser decriptografada apenas com sua chave pública. Como a chave privada é de conhecimento apenas do remetente, por conseguinte, apenas ele pode criptografar a mensagem, o que acarreta a verificação da assinatura.

2.2. Algoritmo de criptografia RSA

O principal algoritmo de criptografia utilizado pela infra-estrutura de chave pública é o RSA [CHOU 02]. O RSA é um algoritmo de encriptação de dados idealizado por Ron Rivest, Adi Shamir e Len Adleman [REVI 77], fundamentado na Teorias Clássica dos Números, sendo o primeiro algoritmo a possibilitar encriptação e assinatura digital.

Seu funcionamento baseia-se na geração de um par de números – as chaves – de forma que uma mensagem criptografada com a primeira chave possa ser apenas decriptografada com a segunda, entretanto, o segundo número não pode ser derivado do primeiro. Esta propriedade assegura que o primeiro número possa ser divulgado a alguém que se pretenda enviar uma mensagem criptografada ao detentor do segundo número, já que apenas essa pessoa pode decriptografar a mensagem. O primeiro número é designado como chave pública, e o segundo como chave privada [CHOU 02].

O algoritmo RSA baseia-se no método onde são encontrados dois números primos de grandes dimensões (e.g. 100 dígitos), esta tarefa é trivil para os computadores, entretanto, conseguir a fatorização do produto de tais números é considerado computacionalmente impossível, em virtude do tempo necessário para realização da tarefa (milhares de anos) [CHOU 02]. De fato, este algoritmo mostra-se computacionalmente robusto com números de tais dimensões, e a sua força é geralmente quantificada com o número de bits utilizados para descrever tais números. Para um número de 100 dígitos são necessários cerca de 350 bits e as implementações atuais superam os 512 e mesmo os 1024 bits.

Para implementar um sistema de assinaturas digitais com RSA, o utilizador que possua uma chave privada *d* poderá assinar uma mensagem (em blocos) *m* com a seguinte expressão [RAVI 77]:

$$s = m^d \bmod n \quad (1)$$

Como se pode deduzir, é difícil descobrir *s* sem o conhecimento de *d*. Portanto, uma assinatura digital definida conforme esta equação é difícil de forjar. Mais, o emissor de *m* não pode negar tê-la emitido, já que mais ninguém poderia ter criado tal assinatura. O receptor recupera a mensagem utilizando a chave pública do emissor, através da expressão [RAVI 77]:

$$s^e = (m^d)^e \bmod n = m \bmod n \quad (2)$$

O receptor consegue validar a assinatura do emissor calculando se mod n. Podemos verificar então que o algoritmo RSA satisfaz os três requisitos necessários de uma assinatura digital.

A assinatura varia dependentemente da mensagem, variando seu tamanho proporcionalmente ao tamanho da mensagem. Para tratar esta situação, utiliza-se um algoritmo de resumo ou checagem de integridade (*digest*), que identifique essa mensagem como única. Geralmente o *digest* de uma mensagem varia alterando um único byte, o que mantém, como conseqüência, uma assinatura diferente de mensagem para mensagem, para um mesmo emissor [CHOU 02].

2.3. Certificados digitais e autoridades certificadoras

Uma arquitetura onde existem diversas chaves compartilhadas, a administração fica onerada e cresce os riscos em relação à proteção das chaves. [HARR 05] Na infra-estrutura de chave pública, surge a figura de uma terceira parte que detém a confiança da comunidade, investigando os indivíduos e sua real identidade no mundo real, correlacionando a identidade à chave publica e verificando se o indivíduo possui a chave privada. [NOAK 01]

Essa terceira parte confiada é chamada de *certification authorities* (CA), **autoridades certificadoras,** e sua função é assinar digitalmente as chaves públicas com a certeza que está associada ao seu responsável. Em virtude dos sistemas de PKC exigirem um considerável poder computacional, novos tipos de gerenciamento de chaves não se provaram úteis comercialmente até 1990, ao tempo em que a implementação da PKI se tornou viável. [NOAK 01]

Para adquirir o certificado digital, que é o arquivo eletrônico assinado por uma autoridade certificadora, o usuário gera um par de chaves assimétricas: uma chave privada e uma requisição de certificado, que deverá ser enviado à autoridade certificadora. Esta por sua vez, verifica a identificação física do solicitante e assina a requisição com sua chave privada, criando assim um certificado digital assinado. O processo de assinatura de um certificado digital por uma autoridade certificadora é detalhado na Figura 2.2.

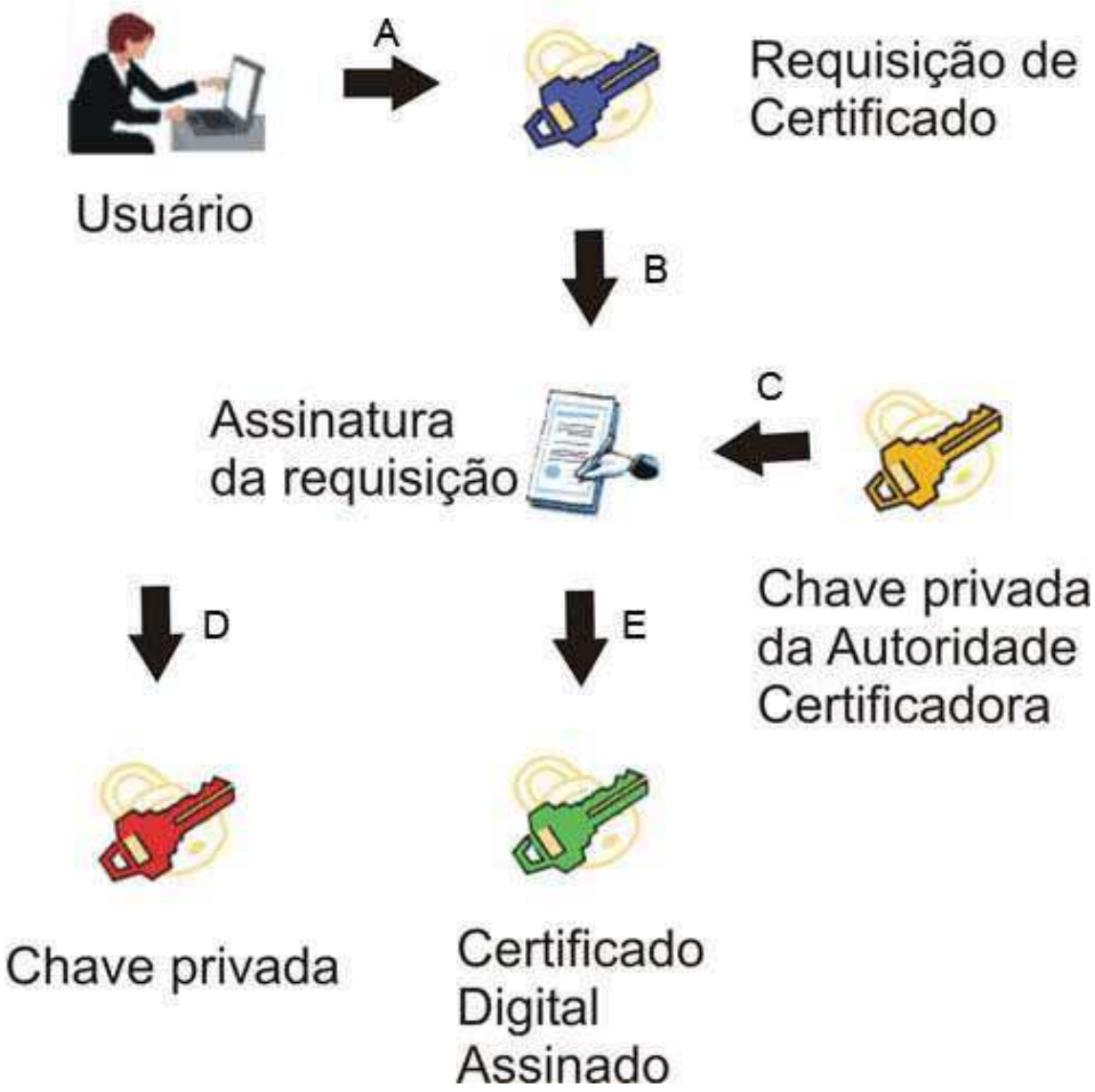

Figura 2.2 – Processo de criação de um certificado digital.

O certificado digital é usado para verificar a entre uma chave pública e o dono da chave (este tipo de certificado também é conhecido como certificado de chave pública). Quando uma autoridade certificadora emite uma um certificado digital, assinado digitalmente com sua chave privada, ela certifica que aquela chave pertence realmente a um determinado indivíduo. Noaks-Fry [NOAK 01] afirma ainda que a quantidade de informação sobre um usuário está embarcada no certificado e varia de acordo com o sistema usado.

Um certificado digital não pode por ele mesmo, habilitar uma transação segura, ou prover autenticação ou não repúdio. [HARR 05] Ele serve para indicar a parte interessada a identidade de uma chave pública, que por sua vez, pode utilizá-la para seus processos de autenticação (as partes são quem realmente dizem que são) e não repúdio (proteção contra uma negação da participação de uma parte em um ato). Harris [HARR 05] salienta ainda que os recursos de não repúdio de um certificado digital não podem prevenir a entidade de repudiar uma comunicação. Ao invés disso, usas-se o certificado digital guardar evidências que podem ser apresentados a uma terceira parte para resolver disputas sobre as ações das entidades envolvidas.

A Figura 2.3 mostra como uma autoridade certificadora atua para verificar a identidade de um usuário.

Figura 2.3 – Processo de verificação de uma assinatura digital.

A mensagem é enviada para o destinatário criptografada com as informações da chave privada do remetente e de seu certificado digital que por sua vez, será utilizado pelo destinatário para verificar com a chave pública da autoridade certificador se o certificado é válido e se o mesmo pertence ao remetente.

Ao receber um arquivo assinado através de uma chave pública e um certificado digital, o destinatário pode compará-lo com a chave pública da autoridade certificadora emissora, verificando assim, a autenticidade da assinatura.

2.4.Composição da Infra-estrutura de Chave Pública (PKI)

2.4.1. Componentes

O termo PKI fui utilizado pela primeira vez em 1980 por um time da Bell Norther Research (hoje Nortel Secure Networks) no Canadá quando estavam desenvolvendo pesquisas na área de PKC. A infra-estrutura de chave pública utiliza elementos como Autoridades Certificadoras (CA), Autoridades Registradoras (RA) e diretórias. O detalhamento das atividades destes objetos é descrita na Tabela 2.1.

Componentes de uma PKI	
Certification Server (CS) ou Servidor de Certificados, também chamado de Autoridade Certificadora.	Uma Autoridade Certificadora, dentro de uma PKI, emite os certificados que forem previamente autenticados. A Autoridade Certificadora pode ser uma organização, um indivíduo, ou uma agência pública ou privada, agindo como uma terceira parte de confiança. Esta confiança é gerada pelo contexto em que sua chave privada, que assina todos os outros certificados, está protegida e pode ser validada. • Esta plataforma cria, emite e assina certificados com sua chave privada. • Também publica certificados e lista de revogações. • Usuários podem ser indivíduos em suas estações de trabalho ou aplicações. Eles fazem uso da chave pública da autoridade certificadora para verificar certificados ou assinaturas oferecidas para checar a

	identidade ou qualquer informação associada.
Registration Authority (RA) ou Autoridade Registradora.	Uma Autoridade Registradora é um componente opcional de uma PKI que provê uma interface entre um usuário e a plataforma emissora do certificado. • A Autoridade Registradora verifica a identidade de uma entidade fim (uma pessoa, função, servidor ou terminal, que deseja requerer um certificado; • Uma Autoridade Certificadora pode servir a várias Autoridades Registradoras. Algumas empresas centralizam a CA e distribui a RA por vários sites e departamentos; • A Autoridade Registradora pode agir de modo on-line ou off-line, dependo da sua relação de atualização com a CA; • Os serviços de Autoridade Registradora geralmente são executados em estações de trabalho, pois suas informações não precisam estar publicamente disponíveis. Diversas funções exercidas de RA podem sem realizadas por outros elementos da PKI.
Diretório	Os serviços de diretórios (e.g. Ldap) são necessários para armazenar os certificados e dados de suporte.
Aplicações com suporte a PKI	Dentro de um arcabouço de PKI, as aplicações necessitam suportar a infra-estrutura de chaves, como por exemplo, os clientes de e-mail, túneis de VPN, navegadores de internet (*browsers*).

Tabela 2.1 – Componentes de uma PKI.

2.4.2. Padrões utilizados

Os padrões que excutam um importante papel na implementação e uso de chaves públicas de criptografia podem ser divididos em quatro grupos (Tabela 2.2), cada um atendendo um seguimento das tecnologias necessárias para manter a PKI. Diversas instituições, grupos e empresas, como a Internet Telecommunication Union (ITU) e a Internet Engineering Task Forces descutem e estabelecem os padrões utilizados, como a criptografia, diretórios, cabeçalhos e outros.

Padrões de uma PKI	
Grupo 1	
O primeiro grupo compara o padrão X preparado pela *International Telecomunication Union* (ITU). Estes padrões são internacionalmente reconhecidos e usados para descrever o diretório e os significados da informação codificada da estrutura de armazenamento.	X.500 - Diretório: visão geral dos conceitos, modelos e serviços; X.509 - Diretório: arcabouço de autenticação; X.509a - Prévia das extensões de certificados (versão 3); X.208 - Especificação do *Abstract Syntax Notation One* (ASN.1); X.209 - Especificação das regras básicas de codificação para o ASN.1.
Grupo 2	
O segundo grupo é o Internet X.509 PKI (PKIX). Muitos grupos do *Internet Engineering Task Force* (IETF) estão produzindo requisições de comentários (RFCs); Entretanto, este grupo desenvolve um notável trabalho no diretório e nos protocolos de acesso. Os padrões desenvolvidos por este grupo definem como requisitar certificados de diretórios através da internet, como lidar com listas de revogação, políticas de certificados e gerenciamento dos formatos das mensagens.	• PKIX 1 (RFC 2459) *Internet X.509 Public Key* *Infrastructure*: X.509 certificado e perfil da lista de revogação (CRL); • PKIX 3 (RFC 2510) *Internet X.509 Public Key* *Infrastructure*: Protocolos de gerencimento de certificados; • LDAPv.2 Protocolos operacionais da Infra-estrutura de chave pública.
Grupo 3	
O terceiro grupo consiste nos padrões da indústria, alinhados ao protocolo de criptografia RSA. Estes são os padrões públicos de criptografia, que estabelecem a camada de criptografia necessária para a troca de informações entre entidades. Estes padrões estão sendo evoluídos e/ou substituídos por outros padrões, como o *Extensible Markup Language* (XML). Este grupo de padrões identifica as maneiras nas quais os objetos criptografados são tratados na autoridade certificadora.	• PKCS#7 sintaxe padrão da mensagem criptografada; • PKCS#10 padrão da requisição do certificado; • PKCS#11 seleção dos tipos de atributos.
Grupo 4	
Uma série de padrões adicionais que descrevem a estrutura da mensagem e protocolos que usam técnicas de segurança.	• S/MIME (RFC 2633): *Secure Multipurpose Internet* *Mail Extensions, version 2* • S-HTTP: *Secure Hypertext Transport Protocol*

	• SSL: *Secure Sockets Layer* • TLS (RFC 2246): *Transport Layer Security* • IPSec 6 (RFC 2401 e outras): *Internet Protocol Security, version 6*

Tabela 2.2 – Padrões utilizados pela PKI.

2.5. Benefícios e riscos da utilização de PKI

2.5.1. Benefícios

Em seu trabalho, Noaks-Fry [NOAK 01] destaca como principais benefícios da utilização da infra-estrutura de chave pública os recursos de segurança da identificação e não-repúdio. A PKI provê o arcabouço necessário para autenticar as partes em uma transação e criptografar a comunicação, garantindo a **segurança da identificação** contra impostores, interceptadores e indivíduos que podem maliciosa ou acidentalmente, alterar documentos ou forjar identidades.

A identificação positiva das partes através das assinaturas oferece um registro de valor legal da transação eletrônica, que dificulta qualquer parte em envolvida a negar um ato realizado – este é o conceito do **não-repúdio**.

Podemos ainda enumerar algumas vantagens administrativas e aplicações da PKI, que facilitam sua utilização e eleva a segurança de todo o sistema: [HARR 05]

- Oferece uma consistente interface de administração;

- Em alguns casos, organizações podem eliminar a necessidade de múltipla autenticação nos sistemas e múltiplas senhas;
- Gerenciamento de usuários pode ser simplificado em virtude que seus direitos de acesso podem ser retirados com uma simples função de revogação.
- Segurança de *e-mails* sob S/MIME (*Secure Multipurpose Internet Mail Extensions*);
- Segurança da propriedade intelectual através de encriptação;
- Gerenciamento digital de ativos;
- *Intranets* e *Extranets* seguras;
- Transações comerciais e financeiras via internet;
- Acessos remotos (VPNs);

Diversas organizações públicas e privadas estão estudando formas de entregar serviços mais rápidos e flexíveis aos seus clientes, particularmente focados na abrangência de mercados mundiais. [NOAK 01] Bancos, agências governamentais, seguros de saúde, companhias farmacêuticas, seguradoras, comércio eletrônico, dentre outras estão se beneficiando da PKI, trazendo mais segurança aos seus negócios.

2.5.2. Riscos

Noaks-Fry [NOAK 01] também sumariza alguns riscos que devem ser controlados para garantir o perfeito funcionamento de uma PKI. Estes riscos estão relacionados à proteção da chave privada da autoridade certificadora, proteção das chaves privadas individuais, interrupção no processamento, necessidade de interoperabilidade e arcabouço legal, explicados com mais detalhes abaixo:

Uma PKI é tão boa quanto à **proteção de sua chave privada**. Todo conhecimento comum de segurança de informações deve ser reforçado, incluindo controle de acesso, segurança física e planos de contingência. Uma chave comprometida acarreta a possibilidade de um atacante assinar certificados falsos, deteriorando a credibilidade de todo o sistema.

A **chave privada do usuário** também deve ser protegida com finalidade de evitar sua utilização por um impostor incutindo responsabilidades ao seu verdadeiro proprietário. Mecanismos como os *smart cards* são direcionados para resolver este problema.

Planos de contingência devem ser desenvolvidos para dirimir **interrupções no processamento**, como cobrir a perda de uma chave privada ou o repositório de chaves ficar inacessível.

A **necessidade por interoperabilidade** significa que todo usuário tem a habilidade dede receber um certificado digital, processá-lo, requerer um certificado correspondente, processa-lo com qualquer informação associada e entender os resultados gerados. Muitos fornecedores de PKI fazem parte de comitês e forças tarefas para estabelecer e aceitar os padrões industriais.

Muitos países ainda não regulamentaram a **legislação da assinatura digital**. Isto pode ocasionar a sua não utilização com valor legal em processos judiciais e outras aplicações.

2.6. Utilização da PKI em dispositivos móveis e estados da arte

Diversas iniciativas foram propostas por vários pesquisadores para se implementar a infra-estrutura de chave pública em dispositivos móveis, como veremos no levantamento da bibliografia pesquisada, descritos e comentados a seguir.

Wolfl em seu trabalho de 2005 [WOLF 05] propõe uma arquitetura de PKI baseada em redes ponto a ponto (*peer-to-peer*). Cada ponto conectado à rede confia em um conjunto de outros pontos, criando uma espécie de "teia de confiança". Esta implementação promove a vantagem de resistência à falha, balanceamento de carga, armazenamento redundante e auto-administração. Em contrapartida, a falta de um repositório centralizado pode acarretar problemas de revogação de chaves e latência de propagação.

Zhou and Haas, em [BING 05] desenharam um esquema seguro de gerenciamento de chaves empregando uma mínima quantidade de criptografia. Este sistema pode tolerar servidores compromissados, mas não descrevem como os nós podem contatar os servidores de modo seguro e eficiente, quando estes estão distribuídos e possuem sistemas de autenticações heterogêneos. Também não descreve como ocorrerá a distribuição das chaves.

Luo and Lu, em [BING O5] criaram um esquema centralizado de gerenciamento chamado URSA. Em seu esquema, todos os nós são servidores. A vantagem desta

arquitetura é a disponibilidade de todo o sistema, por outro lado, esta implementação é insegura para equipamentos sem uma proteção física adequada (o caso dos dispositivos móveis) e o tamanho das chaves geradas, que inviabiliza sua utilização em diversos dispositivos e degradam a banda de comunicação. Também, segundo [BING 05], necessita de uma grande quantidade de configuração nos dispositivos.

Yi et al, em [KAWA 02] proporam um esquema chamado *mobile certificate authority* (MOCA), ou autoridade certificadora móvel. Nesta abordagem, o serviço de certificados é distribuído para os nós MOCA, que são fisicamente mais seguros e poderosos que os outros nós. Neste esquema, um nó pode localizar um nó MOCA de forma randômica, através de caminho mais curto ou baseado numa rota previamente cadastrada em seu cachê. A questão crítica, segundo [KAWA 02] é como os nós podem descobrir qual o caminho mais seguro no estabelecimento do serviço de chaves.

Capkun et al., em [BING 05], considera um esquema totalmente distribuído como uma vantagem em relação à flexibilidade. Entretanto, falta um ponto seguro na estrutura. Muitos certificados necessitam ser gerados e cada nó deve coletar e manter atualizado seu repositório de certificados, tornando proibitivo em alguns dispositivos móveis. O grafo dos certificados, no qual este modelo calça sua teia de confiança, pode não estar fortemente conectado, especialmente em um cenário ad-hoc, onde a indisponibilidade de um nó pode comprometer a arquitetura. Pode também haver um conflito de certificados.

Recentemente, Yi e Kravets em [WOLF 05], desenharam um modelo composto de confiança. Este modelo combina um nó central de confiança com nós distribuídos, como no MOCA de Yi et al [KAWA 02]. Este esquema utiliza os aspectos positivos de dois diferentes sistemas de confiança. Entretanto, existem discussões referentes as

formas de conexão pois alguns mecanismos de autenticação ainda não são tratadas de forma eficiente.

A arquitetura para a utilização da infra-estrutura de chaves pública para dispositivos móveis sofre diversos problemas em virtude da limitação dos dispositivos móveis atualmente disponíveis. Primeiramente, na arquitetura convencional, quando o usuário utiliza certificados digitais, a criptografia utilizada onera a capacidade de processamento dos computadores utilizados, não tão abundantes nos dispositivos móveis. Além disso, a interface de comunicação destes dispositivos utiliza radiofreqüência, que geralmente sofrem de problemas de comunicação. Uma PKI para operar de forma satisfatória deve resolver os seguintes problemas: [KAWA 02]

- Para garantir a interoperabilidade da PKI, todos os certificados precisam estar de acordo com as especificações do padrão X.509;
- Para a autoridade certificadora de uma PKI para dispositivos móveis adquirir as informações necessárias, deve-se reduzir o número de informações que os usuários devam entrar em seus aparelhos;
- Para garantir a persistência de um processo de PKI móvel, em caso de perda da conexão de radiofreqüência, o dispositivo móvel e a autoridade certificadora devem gerenciar o estado da conexão.

Além disso, a capacidade de armazenamento e processamento de alguns dispositivos móveis torna proibitivo a utilização da PKI da maneira que é implementada nos computadores convencionais. Como exemplo, um repositório de certificados das autoridades certificadoras de um navegador de internet padrão possui 680 kilobytes de tamanho (Mozilla – Firefox 2.0), capacidade exorbitante para um aparelho móvel armazenar para apenas um serviço.

Em resumo, o problema da autenticação de dispositivos móveis é tratado de várias formas, com aspectos positivos e negativos, mas existem poucos trabalhos tratando da interoperabilidade com a PKI, que é um padrão reconhecido e utilizado pelo mercado.

2.7. Conclusão

Este capítulo apresentou a fundamentação teórica sobre o assunto investigado nesta dissertação, analisando os riscos e benefícios bem como o estado da arte de outras soluções já desenvolvidas sobre o mesmo tema.

Capítulo 3 – Arquitetura

Neste capítulo descreveremos os requisitos necessários para uma arquitetura que suporte a infra-estrutura de chave pública em dispositivos móveis, propondo uma solução para a o problema e discutindo as tecnologias utilizadas.

3.1. Requisitos necessários

Para a utilização da infra-estrutura de chave pública em dispositivos móveis, a arquitetura deverá ser desenhada de modo a atender os requisitos necessários da PKI e ao mesmo tempo, amenizar as limitações existentes nos dispositivos interligados. Além disso, é desejável que essa arquitetura seja capaz de se interligar com outras arquiteturas e tecnologias, bem como, estar alinhada com padrões de mercado e possuir capacidade de evolução.

Em resumo, a arquitetura proposta deverá atender os seguintes requisitos:

- Ser compatível com o padrão X.509, que define os requisitos aceitos amplamente pelo mercado da infra-estrutura de chaves pública;

- Utilizar uma Autoridade Certificadora (CA) e certificados digitais assinados pela mesma, compatíveis com outras aplicações, como por exemplo, navegadores de internet;
- Possuir a capacidade de gerar sementes criptográficas para outras formas de comunicação segura, como o SSL e TLS;
- Funcionar em dispositivos móveis, como celulares e PDAs, com pouca capacidade de processamento e armazenamento;
- Usar protocolos de comunicação abertos, que possuam ou possam ser implementados, mecanismos de segurança para o transporte de dados e outros riscos;

Além dos requisitos mencionados, seria desejável que a arquitetura também fornece mecanismos para aprimoramento da computação pervasiva, não limitando as características dos dispositivos móveis (mobilidade e comunicação). Seria interessante, no entanto, que a arquitetura possuísse recursos de resiliência, para não depender de um único provedor de serviços e também a possibilidade de atuar ou conectar-se com redes IP, para aumentar sua abrangência de atuação.

3.2. Arquitetura proposta

A infra-estrutura de chave pública, em seu padrão X.509, disponibiliza todas as chaves públicas das autoridades certificadoras para que os clientes e aplicações possam verificar uma assinatura eletrônica. Esta implementação é realizada de várias formas:

armazenamento de chaves em diretórios LDAP, banco de dados públicos ou locais, implementações do tipo *stand-alone*, dentre outras. [HARR 05]

Os navegadores de internet em transações bancárias, uma das aplicações mais populares da PKI, possuem um diretório local contendo chaves públicas de diversas AC e recursos para a instalação de outras chaves que surgem. Este diretório possui em média um tamanho de 680 kilobytes. Esta forma de implementação se torna inviável em dispositivos móveis, devido ainda sua escassez de recursos. Tomando como exemplo *smartphone* modelo 6630, do fabricante Nokia [NOKI 06], um dos mais modernos atualmente, possui 10MB para armazenar arquivos e programas, uma implementação deste diretório iria consumir cerca de 7% de sua de capacidade total de armazenamento, proporcionalmente, 5.6 gigabytes em um microcomputador (disco rígido de 80 gigabytes).

Outra grande limitação que impede esta implementação decorre na necessidade de processamento criptográfico das chaves recebidas pelo cliente. Os dispositivos móveis ainda possuem restrita capacidade de processamento o que pode onerar seu desempenho.

Em virtude dessas limitações, é proposta a utilização de um serviço de armazenamento e checagem de chaves, montando em uma estrutura de cliente-servidor, acessado pelos dispositivos móveis e possuindo a confiança dos usuários.

Este servidor exercerá a função de receber as chaves enviadas pelos dispositivos móveis, verificar sua autenticidade comparando com um repositório das chaves públicas das autoridades certificadoras e retornando ao requisitante o resultado da consulta. Como ocorre uma relação de confiança entre o cliente e o servidor, estes deverão estar conectados e confiarem fortemente na arquitetura.

O diagrama de eventos da Figura 3.1 mostra o funcionamento da arquitetura em uma possível aplicação. Neste caso, um dispositivo A deseja iniciar uma sessão de comunicação com um dispositivo B. O dispositivo A envia sua solicitação juntamente com uma mensagem assinada por ele e B recebe a mensagem e repassa para o servidor, que irá verificar a autenticidade da assinatura junto aos certificados das autoridades certificadoras, retornando o resultado da consulta para B, que decide se inicia ou não a sessão.

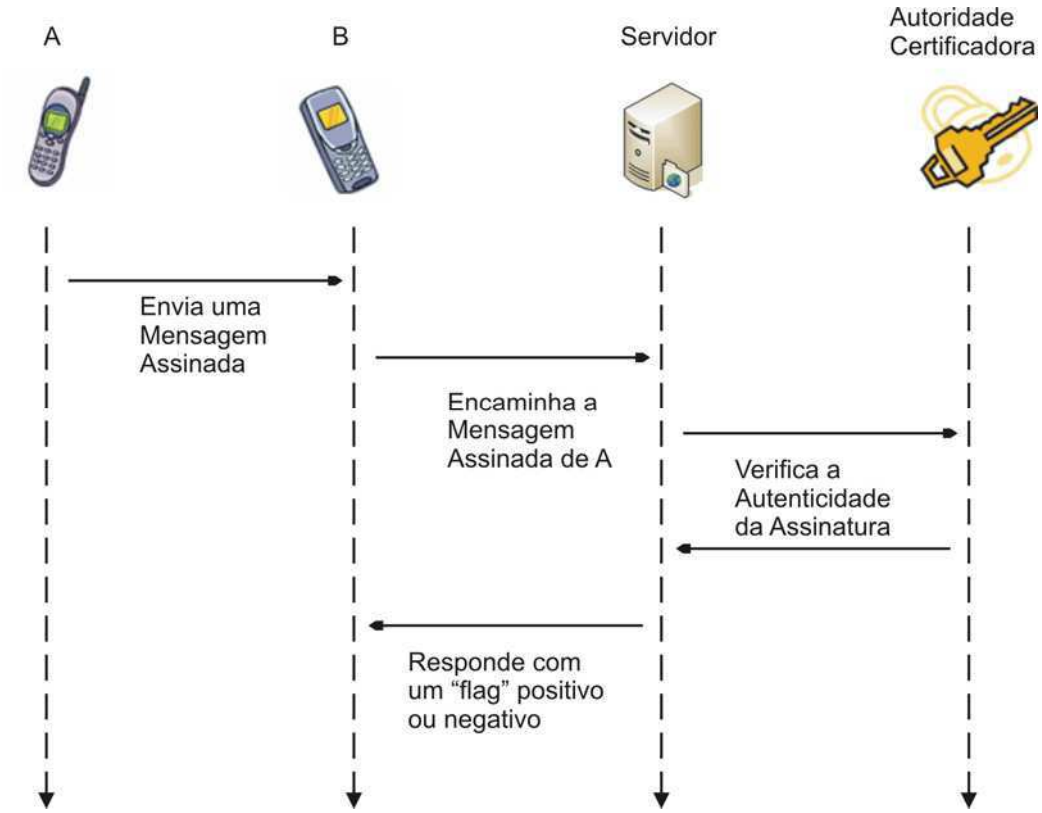

Figura 3.1 – Exemplo de uma aplicação da arquitetura proposta

A arquitetura também poderá se beneficiar do envio de chaves no início da sessão para verificar a autenticidade de seus nós e iniciar um canal de comunicação criptografado, como SSL, dentre outras funcionalidades.

3.3. Topologia

Para o funcionamento da arquitetura, a mesma deverá possuir pelo menos um dispositivo móvel comunicando-se com um servidor, que por sua vez, consiga fazer a checagem de assinaturas em um repositório de chaves. O esquema da Figura 3.2 mostra uma proposta desta topologia.

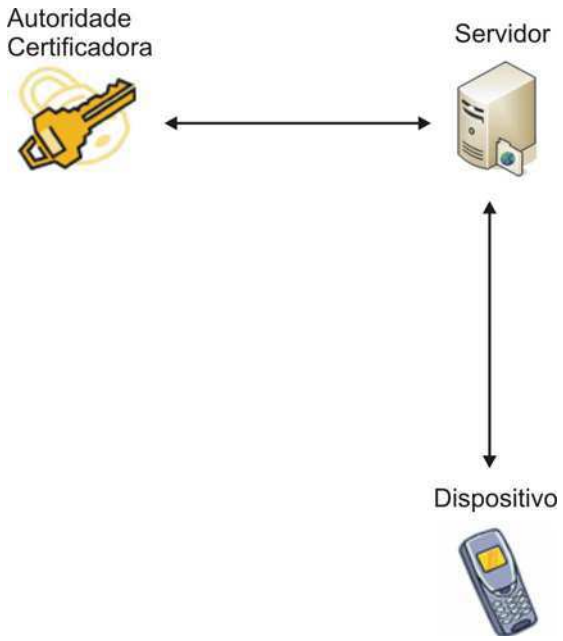

Figura 3.2 – Proposta de topologia para a arquitetura

3.4. Tecnologias utilizadas

A implementação da arquitetura proposta necessita alguns requisitos que devem ser sanados pelas tecnologias utilizadas. Como exige certo grau de robustez e confiabilidade, as tecnologias de comunicação exercem função fundamental, pois os

objetos devem confiar na infra-estrutura, além de possuir interoperabilidade com outros serviços e aplicativos. O servidor deve ser desenvolvido em uma linguagem que possua suporte tanto para a infra-estrutura de chave pública como para o protocolo de comunicação utilizado.

Como o envio do certificado ocorre no início da comunicação, o protocolo de comunicação deve possuir controles de sessão eficientes, sendo também flexível para suportar alterações e encapsular diferentes tipos de conteúdos. Além disso, deve ser compatível com dispositivos móveis e aceito pelo mercado. Dentre os protocolos de comunicação existentes que suportam comunicação IP (*Internet Protocol*), necessário para a interoperabilidade da arquitetura, destacamos o protocolo SIP (*Session Initiation Protocol*) que preenche os requisitos exigidos pela arquitetura. O protocolo SIP atua como um facilitador para estabelecimento de sessões, ponto-a-ponto ou através de servidores, com a flexibilidade de ser utilizado em conjunto com outros protocolos, como por exemplo, os protocolos para aplicações em tempo real e criação de túneis SSL. [CAMA02] Além disso, por ser um padrão aberto e baseado em texto puro, novos cabeçalhos podem ser criados e alterados. A pilha SIP já foi portada para dispositivos móveis e várias linguagens de programação já possuem suporte, como o Java.

Originalmente as implementações de protocolos de sinalização eram proprietárias. Com o passar do tempo, o IETF (Internet Engineering Task Force) e o ITU-T (International Telecommunications Union – Telecommunications Standardization Sector) estabeleceram protocolos padrões para sinalização, possibilitando a implantação de soluções abertas, independentes de fabricante e integradas aos diversos serviços disponíveis nas redes IP. [CAMA02]

Atualmente os principais protocolos de sinalização, além do SIP são o H.323, Media Gateway Control Protocol (MGCP), H.248/ Megaco, MGCP e Megaco.

Os protocolos MGCP e Megaco baseiam-se numa arquitetura centralizada, na qual os dispositivos da ponta (os teerminais) são dotados de capacidade limitada, tornando-os mais baratos e simples de serem construídos. Em uma aplicação típica de voz sobre IP, toda a inteligência está no núcleo da rede, em equipamentos chamados Call Agents, de forma muito parecida com o que existe hoje na rede telefônica convencional (PSTN-Public Switched Telephone Network), uma vez que os terminais (endpoints) são inteiramente dependentes das instruções, inteligência e controle providos pelos Call Agents. O MGCP foi desenvolvido pelo IETF (órgão que padroniza protocolos da Internet), e o Megaco é resultado de um desenvolvimento conjunto entre o IETF e o ITU (órgão de padronização de protocolos de telecomunicações), resultando num melhoramento do MGCP. [CAMA02]

O protocolo H.323 baseia-se no modelo distribuído, onde a inteligência encontra-se nos equipamentos da ponta, mas também podem ser empregados no modelo centralizado em presença dos respectivos Call Agents, cuja função é prover serviços complementares (chamada em espera, transferência de chamadas, entre outros). A recomendação H.323, estabelecida pelo ITU-T, é composta por um conjunto de protocolos que definem os procedimentos para o estabelecimento, a manutenção e a desconexão de chamadas telefônicas no ambiente de redes locais.

Atualmente há poucas implementações baseadas em SIP, as melhores práticas para a instalação e configuração de ambientes SIP estão em processo de elaboração, o que tem prejudicado a adoção deste protocolo em larga escala. Em contra-partida, sua simplicidade, escalabilidade e a utilização do modelo baseado na tecnologia de Internet, tornam o SIP muito atrativo frente ao H.323. [CAMA02]

O protocolo SIP foi escolhido como o protocolo de sinalização por diversas instituições responsáveis por discutir e estabelecer os padrões da comunicação móvel,

como o IETF PINT *working group*, 3GPP (para as redes sem-fio de terceira geração), International Packet Communications Consortium, IMTC e ETSI Tiphon (integração entre SIP e H.323). [SCHU 06]

3.4.1. Protocolo SIP (Session Initiation Protocol)

O Session Initiaton Protocol (SIP) é um protocolo de controle (sinalização) da camada de aplicação desenhado para criar, modificar e terminar sessões de comunicações. [LIN 05] Estas sessões incluem telefonia via internet, distribuição de serviços multimídia, mensagens instantâneas e outros.

Dentro de suas características técnicas, destacam-se a existência de somente seis métodos (REGISTER, INVITE, ACK, OPTIONS, BYE e CANCEL), reduzindo o nível de complexidade. Com independe do protocolo de transporte, pode ser usado com UDP, TCP, ATM e outros. Além disso, seu funcionamento baseado em texto puro diminui a sobrecarga do sistema. [LIN 05].

A arquitetura do protocolo SIP define algumas entidades necessárias: Um **Agente do Usuário** (UA), que é a interface que interage com o usuário final, **Servidores de Proxy**, que manipulam as conexões recebidas e redirecionando-as (se necessário) e **Registradores**, que identifica o usuário e sua localização. A Figura 3.3 mostra a atuação do SIP em uma sessão de comunicação. [CAMA 02]

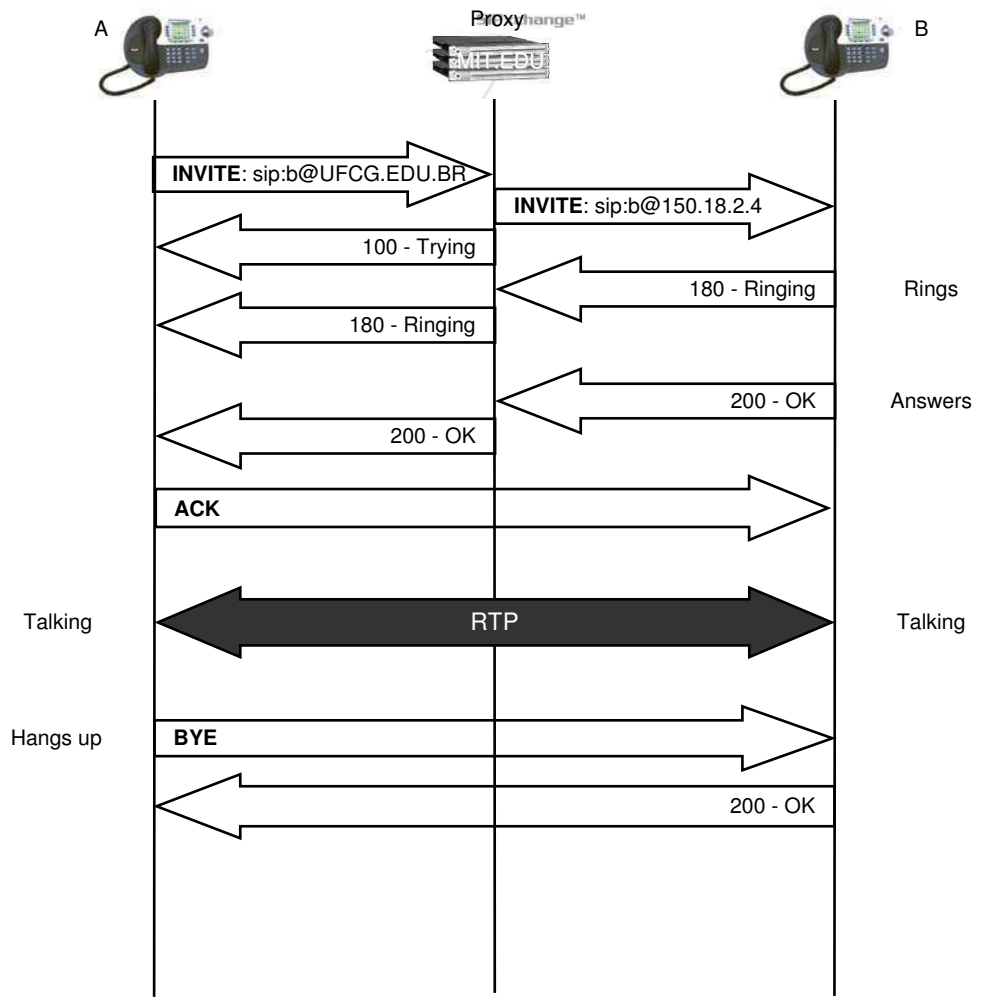

Figura 3.3 – Exemplo da atuação do SIP em uma sessão de comunicação

Camarillo [CAMA 02] descreve várias vantagens do protocolo SIP. Estas vantagens abordam o SIP como parte de kit de ferramentas da IETF (*Internet Engineering Task Force*), separação entre estabelecimento e descrição da sessão, protocolo ponto-a-ponto, interoperabilidade, escalabilidade e sua utilização como plataforma para criação de serviços.

Como uma ferramenta do IETF, o SIP executa suas funções e utiliza outros mecanismos já estabelecidos na internet para execução de serviços adicionais. Isto provê grande flexibilidade aos sistemas poderem utilizar o SIP em conjunto com outros protocolos da internet de maneira modular.

O SIP difere claramente o estabelecimento da sessão e sua descrição. Como parte do estabelecimento da sessão, o SIP provê a conectividade localizando os usuários, mas não definindo o tipo ou descrevendo como será a sessão. Esta distinção faz do SIP extremamente cooperativo, podendo ser usado, por exemplo, com o protocolo SDP (*Session Descrition Protocol*) para estabelecer sessões VoIP.

Os grupos de trabalho da IETF acreditam que os protocolos ponto-a-ponto são melhores para prover serviços ponto-a-ponto e que IP é um protocolo com esta característica. IP provê conectividade entre pontos separados por uma rede de roteadores, estes roteadores enviam os datagramas da forma mais eficiente possível. Similarmente, SIP provê conectividade entre usuários através de servidores SIP. Estes servidores possuem a tarefa de encaminhar as requisições SIP para os Agentes do Usuário ou outro servidor SIP. Como não processam as descrições da mensagem, estes servidores não sofrem sobrecarga, tornando eficiente o protocolo.

O protocolo SIP está desenhado para que qualquer protocolo possa se beneficiar de seus métodos de negociação de sessões. Agentes de Usuários (UA) sofisticados podem estabelecer sessões com UAs rudimentares. Todas as extensões SIP são modulares e desenhadas para serem negociadas individualmente. Pode-se utilizar um conjunto de extensões para uma sessão inicial e outra completamente diferente para próxima. As negociações modulares garantem a real interoperabilidade entre todos os usuários SIP da rede.

O a inteligência do protocolo SIP está no ponto do usuário final, que necessita gravar informações durante a sessão. Uma vez estabelecida uma sessão de comunicação, esta não precisa mais da assistência do servidor. Os servidores SIP não precisam monitorar a sinalização durante as sessões, com isso, eles funcionam em modo

stateless, manipulando um grande número de sessões e aumentando a escalabilidade da arquitetura.

A utilização como uma plataforma de criação de serviços é visto por Camarillo [CAMA 02] como o mais importante recurso do SIP. O reuso de componentes do SIP torna (deliberadamente) torna possível a utilização de muitos componentes da internet por outras aplicações. Isto faz do SIP o protocolo ideal para combinar diferentes serviços para os usuários. Por exemplo, podem-se combinar serviços de HTTP e SMTP (páginas e correio) com serviços multimídia.

3.4.2. Segurança da arquitetura

Outros aspectos de segurança, como a confidencialidade, integridade e disponibilidade também necessitam ser tratados para viabilizar a arquitetura. Neste contexto o protocolo SIP oferece vários mecanismos para a proteção contra muitos tipos de ataques e suporte a outras tecnologias de segurança, como o uso de certificados digitas para a autenticação, da mesma maneira que os navegadores e servidores de páginas da internet, bem como criptografia dos dados trafegados. Os controles de algumas ameaças já tratadas no protocolo SIP estão descritas na Tabela 3.1: [SINN 06]

AMEAÇA	DESCRIÇÃO	CONTROLE	MECANISMO
Seqüestro de sessão	Redirecionamento de uma sessão destinada a uma pessoa	Autenticação da sinalização, identificação	*Hash*, Identificação avançada, TLS
Violação de mensagens	Uma terceira parte lê e grava troca de mensagens de outros usuários	Confidencialidade do SIP	TLS e MSRP Seguro

Negação de serviço	Uma mensagem não alcança seu destinatário	Gerenciamento de tráfego de IP, SIP e RTP	Vários mecanismos
Interrupção de sessão	Uma mensagem de um usuário para outro é apagada ou bloqueada	Integridade	SIP Seguro (SSIP)

Tabela 3.1 – Mecanismos de segurança do SIP

O SIP também pode utilizar vários mecanismos de autenticação, como por exemplo, o *hash* HTTP, descrito na sessão 22 do RFC 3261 [ROSE 02]. Este mecanismo utiliza uma maneira simples de desafiar um objeto de seu nome de usuário e senha. Estas informações são enviadas em um *hash* MD5, garantindo a confidencialidade e integridade.

3.4.3. TLS (*Transport Layer Security*)

Para reforçar a interoperabilidade com outros serviços da internet, a arquitetura deverá disponibilizar os métodos de troca de chaves criptográficas utilizados por outras implementações, como os navegadores de internet. O protocolo utilizado por estas aplicações é o TLS (*Transport Layer Security*).[LIN 05]

O TLS, sucessor do SSL (*Secure Sockets Layer*), é um protocolo criptográfico que foi idealizado para implementar **autenticação** e **privacidade** nas comunicações na internet como navegação, mensageria e transferência de dados.

Seu funcionamento baseia-se no método de troca de chaves. [HARR 05] Quando um cliente acessa um servidor, ele envia uma mensagem inicial com um conjunto de protocolos de criptografia que ele suporta. O servidor por sua vez, escolhe os

parâmetros de conexão e oferece seu certificado digital ao cliente. Cliente verifica a identidade do servidor e negociam uma chave secreta comum para criptografar os dados transacionados na sessão.

A arquitetura proposta neste trabalho implementa os recursos de autenticação do TLS em conjunto com a Infra-estrutura de chaves pública (PKI).

3.5. Conclusão

Neste capítulo descrevemos os requisitos necessários para uma arquitetura que suporte a infra-estrutura de chave pública em dispositivos móveis. Foi proposta uma solução para a implementação e discutida as tecnologias a serem utilizadas, com destaque ao protocolo de SIP, que faz a iniciação da sessão de comunicação.

Capítulo 4 – Estudo de Caso

Neste capítulo será descrito uma implementação da arquitetura proposta, simulando uma aplicação utilizando dispositivos móveis de modo a se beneficiar da infra-estrutura de chave pública. Serão abordados os casos de sucesso e insucesso bem como uma análise dos resultados obtidos.

4.1. Metodologia

Neste estudo de caso, será implementada a arquitetura proposta simulando a uma aplicação de comunicação entre dois dispositivos móveis, iniciando uma sessão de troca de informações e verificando a identidade do solicitante no servidor de chaves, repassando ao destinatário. Neste tudo de caso serão abordados os assuntos referentes ao início de sessão de troca de dados, independente do tipo de dado que será trafegado e a validação de chaves em um servidor, que verificará as chaves em uma infra-estrutura de chave pública. Apesar de ser uma implementação simples, a simulação poderá ser portada para ambientes mais complexos e dispositivos reais.

No estudo de caso, um dispositivo móvel A tentará iniciar uma sessão de comunicação com um dispositivo B. Estes dispositivos farão essa transação através do protocolo SIP. Ambos já estão registrados no servidor SIP que não requer autenticação e funcionará como um servidor público, apenas realizado o serviço de Proxy. No momento em que o dispositivo A disparar a tentativa de conexão, enviará seu certificado digital e um arquivo assinado com sua chave privada juntamente com a mensagem de início de sessão. O servidor SIP que está realizando o gerenciamento da sessão irá capturar o certificado digita e verificará sua autenticidade da assinatura frente as informações de uma Autoridade Certificadora. Feito isso, repassará para o dispositivo B o resultado desta análise, juntamente com a solicitação de conexão. O resultado será exibido na tela do dispositivo B, que decidirá se inicia ou não a sessão.

A Figura 4.1 mostra um diagrama dos eventos que ocorrerão no estudo de caso e as entidades participantes:

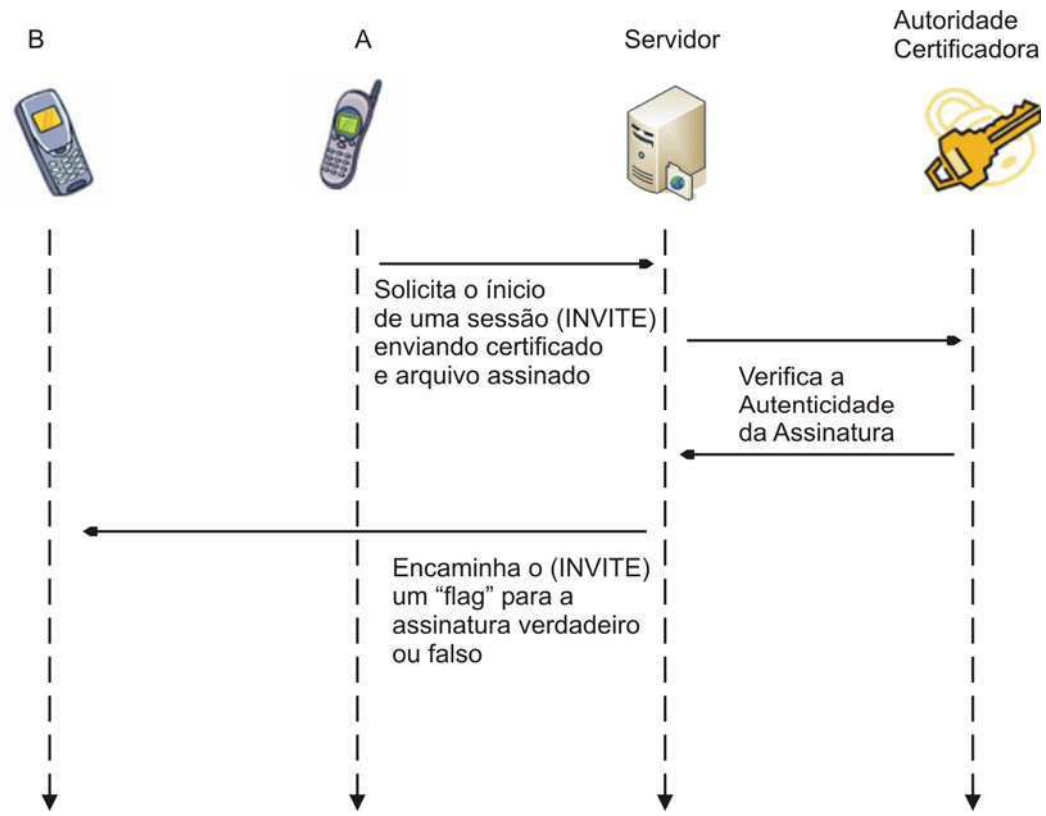

Figura 4.1 – Eventos do estudo de caso

Para a realização do experimento, deveremos disponibilizar as entidades envolvidas, implementado as funções necessárias, bem como a geração das chaves criptográficas para a verificação.

4.2. Autoridade certificadora (AC)

Para a assinatura de das chaves públicas dos usuários e criação da chave da chave de verificação devemos implementar uma Autoridade Certificadora, que funcionará como a terceira parte de confiança.

Para a criação da AC iremos utilizar o software OpenSSL[1] versão 0.9.8d. executado sobre o sistema operacional Linux Debian, versão *testing etch beta release 3*. O OpenSSL é um software de código aberto que possui todas as ferramentas necessárias para a criação e manutenção de uma autoridade certificadora, dentre outros recursos. [XENI 00] Este software é usado por diversas AC públicas pelo mundo e também em arquiteturas privadas.

Primeiramente, deve ser criado o par de chaves do tipo RSA, utilizando a criptografia 3DES. Para o estudo de casos, utilizaremos uma chave de 1024 bits de tamanho. A chave pública será auto-assinada pela chave privada da AC e poderá ser distribuída e instalada nos repositórios públicos. A autoridade certificadora também assinará os certificados digitais dos usuários dos dispositivos móveis. Os comandos para geração e assinatura de chaves estão reproduzidos abaixo.

```
(1) openssl genrsa -des3 -out ca.key 1024
(2) openssl req -new -x509 -days 365 -key ca.key -out ca.crt
(3) openssl genrsa -des3 -out sippya.key 1024
(4) openssl req -new -key sippya.key -out sippya.csr
(5) openssl ca -infiles sippya.csr -out sippya.crt
```

O comando (1) cria a chave privada da autoridade certificadora (AC). Esta chave é do tipo RSA e possui 1024 bits de criptografia *triple des* (3DES). Com a chave gerada, o comando (2) cria um certificado auto-assinado da autoridade certificadora. Este certificado é utilizado para verificar os certificados emitidos pela AC.

[1] <http://www.openssl.org>

O comando (3) cria a chave privada para o usuário SIPPYA, que será o usuário do dispositivo móvel do nosso estudo de caso. Por sua vez, o comando (4) emite uma solicitação de certificado para uma autoridade certificadora. A autoridade certificadora, utilizando sua chave privada, assina com o comando (5) a requisição de certificado de SIPPYA, criando o certificado digital.

O certificado digital criado, por ser inteiramente compatível com o padrão X.509, pode ser utilizado por outras aplicações que suportam a Infra-estrutura de Chave Pública, como por exemplo, os navegadores de internet. A Figura 4.2 mostra o certificado criado incluso no diretório de certificados do navegador de internet Firefox, versão 2.0.

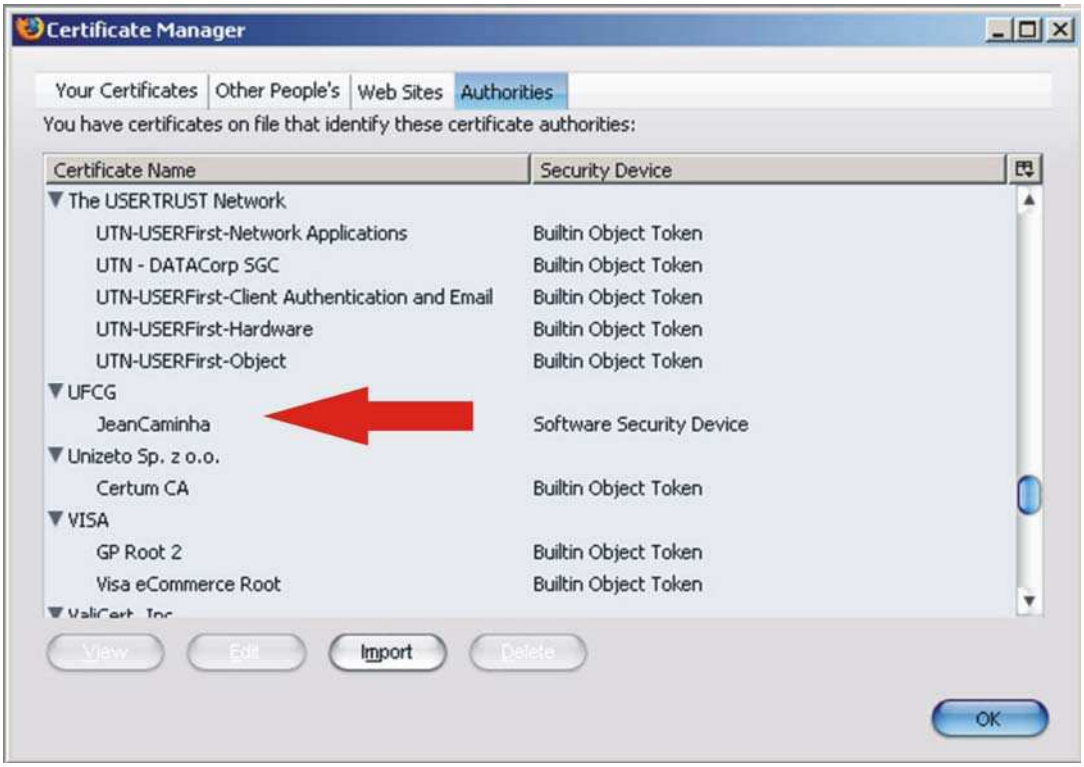

Figura 4.2 Certificado incluso no diretório de certificados de uma aplicação

4.3. Servidor de chaves

O servidor de chaves na arquitetura possui a função de receber a solicitação de verificação de identidade de um dispositivo móvel, checar a validade da assinatura e responder ao solicitante. Também pode ocorrer, no caso da implementação proposta, atuar de modo ativo, verificando a identificação durante o inicio da sessão.

Para implementação do servidor de chaves, acrescentamos a funcionalidade de verificação de chaves no servidor de proxy SIP. O Servidor utilizado foi o JAIN-SIP[2] proxy, versão 1.2, desenvolvido na linguagem Java pelo National Institute of Standards and Technlogy (NIST), dos Estados Unidos, para o projeto *IP telephony*.

Este servidor possui as principais funcionalidades do protocolo SIP e implementadas além de possuir uma excelente interface de rastreamento das conexões.

O código abaixo foi acrescentado ao método de repasse da solicitação de sessão (INVITE) do servidor SIP, para receber a chave pública e um arquivo assinado e fazer a verificação.

```
private boolean validadeCertification(Request request) throws InvalidKeyException, NoSuchAlgorithmException, SignatureException, InvalidKeySpecE
    ExtensionHeaderImpl ownerID = (ExtensionHeaderImpl)request.getHeader("owner-id");
    ExtensionHeaderImpl ownerPublicKey = (ExtensionHeaderImpl)request.getHeader("owner-public-Key");
    ExtensionHeaderImpl ownerEncrypted = (ExtensionHeaderImpl)request.getHeader("owner-encrypted");

    byte[] data = ownerID.getValue().getBytes();
    byte[] sigData = ownerEncrypted.getValue().getBytes();
    PublicKey publicKey = getPublicKey(ownerPublicKey.getValue());
    return this.verify(publicKey,data,sigData);
}

private PublicKey getPublicKey(String value) throws IOException, NoSuchAlgorithmException, InvalidKeySpecException {
    ByteArrayInputStream in = new ByteArrayInputStream(value.getBytes());
    byte[] data = new byte[in.available()];
    in.read(data);
    in.close();

    X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec(data);

    KeyFactory factory = KeyFactory.getInstance("RSA");
    PublicKey pubKey = factory.generatePublic(pubKeySpec);

    return pubKey;
}
```

[2] <http://snad.ncsl.nist.gov/proj/iptel/>

A classe validateCertification recebe as informações referentes ao certificado digital do cliente e o arquivo assinado e realiza a verificação de autenticidade. O resultado desta análise é o retorno de uma variável boleana (*true* ou *false*). A implementação completa deste algoritmo encontra-se no Anexo A desta dissertação

4.4. Dispositivos móveis

Conforme o requisito do projeto do estudo de casos será efetuado a tentativa de início de sessão entre dois dispositivos móveis, utilizado o protocolo SIP e um servidor de verificação de chaves.

O software para os dispositivos móveis foi implementado na linguagem Java e utiliza a biblioteca *Connected Limited Device Configuration* (CLDC)[3], versão 2.5, que define o conjunto base das instruções para criação de aplicativos e utilização da máquina virtual para dispositivos com memória, poder de processamento e capacidade gráfica limitadas.

Primeiramente, foi gerado um arquivo assinado pela chave privada de SIPPYA. Os dados constantes deste arquivo não são relevantes, pois o necessário é a assinatura que eles irão receber. Foi incluso neste arquivo algumas informações sobre usuário, que em outras implementações, pode ajudar a diminuir ainda mais o volume de dados enviado. A finalidade principal deste arquivo é ser verificado via o certificado digital de SIPPYA.

---

[3] <http://java.sun.com/products/cldc/>

A implementação foi desenvolvida modificando o programa de demonstração do protocolo SIP, GoSIP, que acompanha a biblioteca CLDC e a codificação completa encontra-se no Anexo B desta dissertação. Para efeito de simulação, os arquivos foram inclusos dentro do código fonte do dispositivo móvel.

Em seguida, foi modificado o método INVITE da pilha SIP, para carregar as informações referentes ao certificado digital (chave pública) e o arquivo assinado com a chave privada, para que seja efetuada a verificação pelo Servidor de chaves.

O código abaixo mostra as modificações efetuadas na pilha do protocolo SIP, que será enviada ao servidor de chaves.

```
public void addInformation() {
    this.addAdditionalInformation("owner-id","sippy.a@pkisip.br");
    this.addAdditionalInformation("owner-id-at-commonName","sippy.a");
    this.addAdditionalInformation("owner-id-at-organizationalUnitName","PkiSip");
    this.addAdditionalInformation("owner-id-at-organizationName","UFCG");
    this.addAdditionalInformation("owner-id-at-localityName","Campina Grande");
    this.addAdditionalInformation("ca-id","jcaminha@gmail.com");
    this.addAdditionalInformation("ca-id-at-commonName","JeanCaminha");
    this.addAdditionalInformation("ca-id-at-organizationalUnitName","PkiSip");
    this.addAdditionalInformation("ca-id-at-organizationName","UFCG");
    this.addAdditionalInformation("ca-id-at-localityName","CampinaGrande");
    this.addAdditionalInformation("cca-id-at-stateOrProvinceName","Paraiba");
    this.addAdditionalInformation("ca-id-at-countryName","BR");
    this.addAdditionalInformation("owner-public-Key","MIIEnjCCA4agAwIBAgIJAIohbW37CLeOMA0GCSqGSIb3DQEBBQUAMIGQMQswCQYD");
    this.addAdditionalInformation("owner-encrypted","VQQGEwJCUjEQMA4GA1UECBMHUGFyYWliYTEWMBQGA1UEBxMNQ2FtcGluYUdyYW5k");

}
```

O método INVITE do protocolo SIP é bastante flexível e permite a inclusão de vários dados. Neste caso, foi enviado as principais informações do certificado digital de SIPPYA, como a identificação da Autoridade Certificadora além de sua chave pública.

4.5. Verificação do funcionamento

A verificação do funcionamento será efetuada em duas etapas. Primeiramente, iremos verificar um caso de sucesso, onde o certificado e o arquivo assinado pertencem ao usuário do dispositivo e em seguida, alteraremos uma informação do certificado para invalidá-lo, para testar uma identidade negativa.

Nos dois casos, um usuário chamado SIPPYA irá tentar iniciar uma comunicação com o usuário SIPPYB, ambos já se encontram registrados no Proxy SIP. O usuário SIPPYA irá enviar juntamente com a solicitação de INVITE, um arquivo assinado com sua chave privada, previamente instalado no dispositivo e seu certificado. O servidor de chaves, que no estudo de caso também é o Proxy SIP, receberá as informações, fará a checagem da assinatura digital e do certificado digital, encaminhado a solicitação original, juntamente com o resultado da verificação. Este resultado será mostrado no visor do dispositivo móvel SIPPYB, que decidirá se inicia ou não a comunicação.

A Figura 4.3 mostra os dispositivos registrados no servidor Proxy, prontos para iniciarem uma sessão de comunicação.

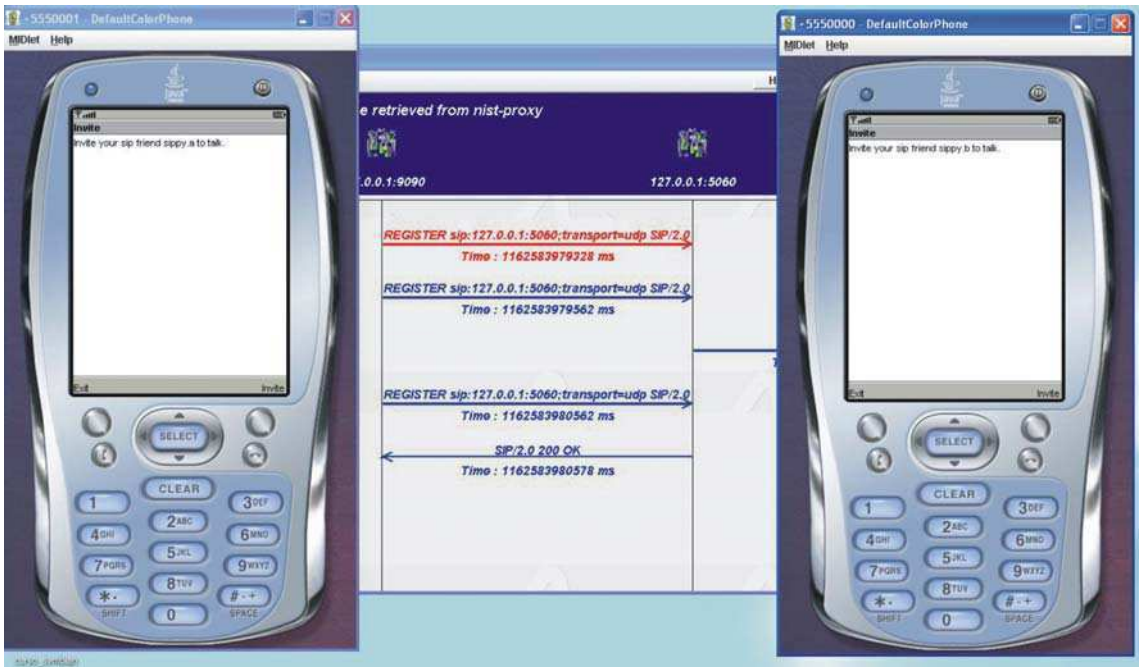

Figura 4.3 Dispositivos registrados no servidor Proxy SIP

4.5.1. Identidade positiva

O dispositivo SIPPYA possui instalado um arquivo assinado com a chave privada e seu certificado digital, que contém sua chave pública. SIPPYA tentará iniciar uma sessão de comunicação com SIPPYB conforme demonstrado no item A da Figura 4.4. No item B da Figura, podemos verificar a captura do pacote enviado no momento da conexão onde acontece o evento.

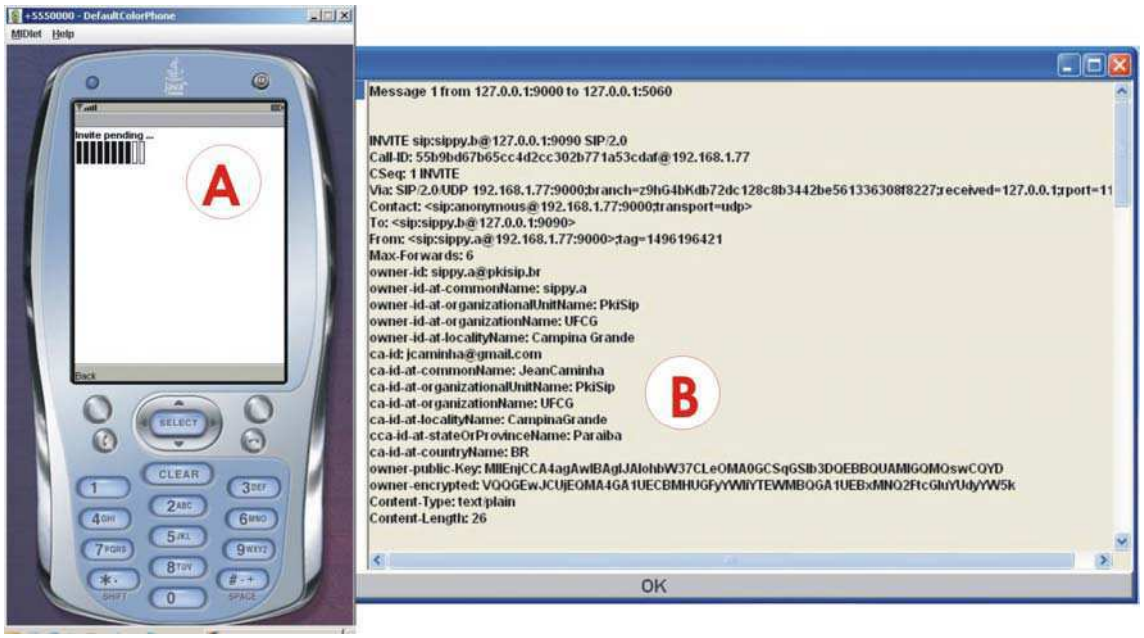

Figura 4.4 Solicitação de início de comunicação pelo dispositivo SIPPYA

O Servidor de chaves, que também é o servidor de Proxy, captura a comunicação, processa o certificado digital e a assinatura digital, encaminhando o resultado da verificação para o SIPPYB, conforme demonstrado no detalhe A da Figura 4.5.

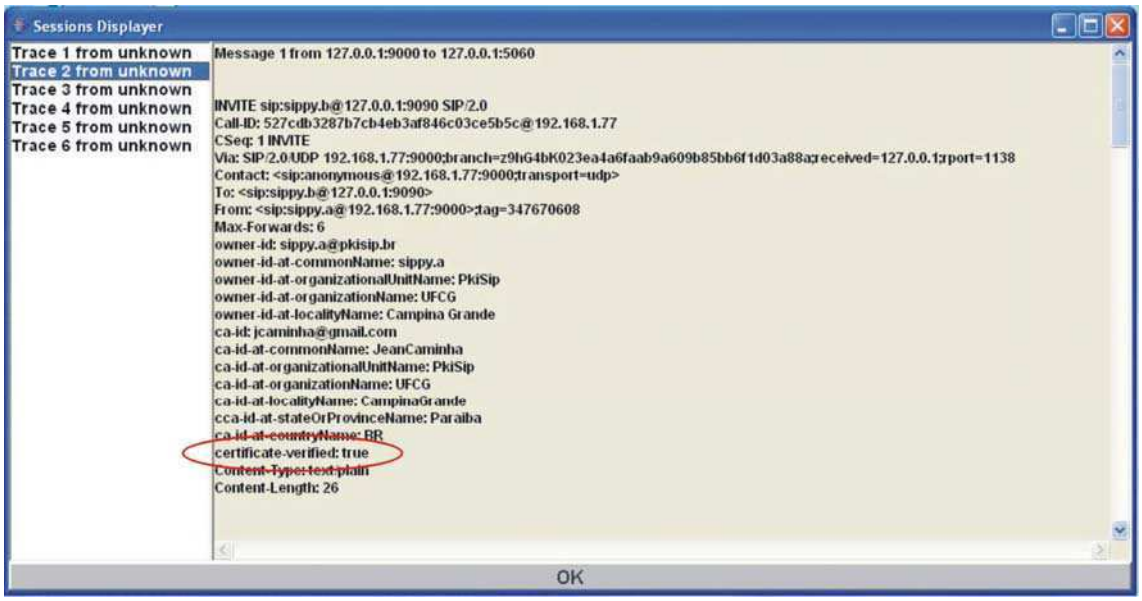

Figura 4.5 Envio do resultado da comunicação para o dispositivo móvel SIPPYB

O dispositivo móvel SIPPYB, recebe o resultado da checagem e exibe para seu usuário, que decide se inicia a comunicação. A mensagem mostrada pode ser verificada na Figura 4.6.

Figura 4.6 Confirmação da identidade de SIPPYA

4.5.2. Identidade negativa

Para verificarmos o funcionamento da arquitetura no caso de uma identidade irregular, efetuaremos uma alteração no arquivo assinado e instalado no dispositivo SIPPYA, de modo a simular uma identidade inválida. A alteração realizada pode ser vista na Figura 4.7.

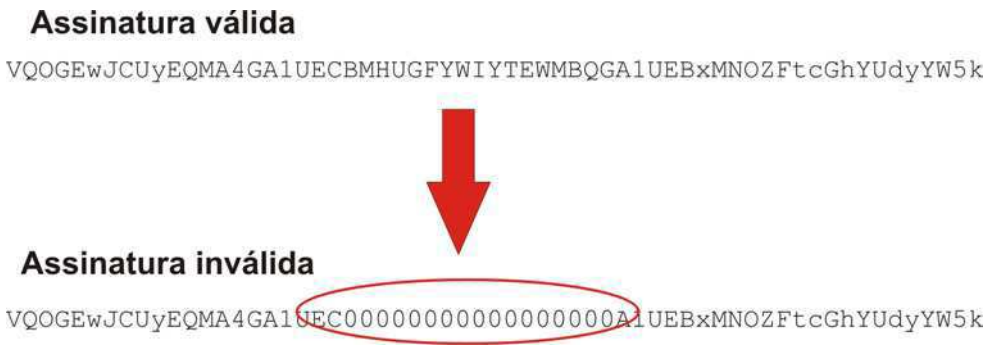

Figura 4.7 Alteração do arquivo assinado

Da mesma forma como ocorreu na verificação positiva, mostrado na Figura 4.5, o arquivo inválido foi enviado juntamente com a solicitação de sessão, para processamento pelo servidor. Este, por sua vez, não atestou a identidade de SIPPYA, comunicando o resultado para o dispositivo SIPPYB, conforme detalhe da captura do protocolo enviado, mostrado na Figura 4.8.

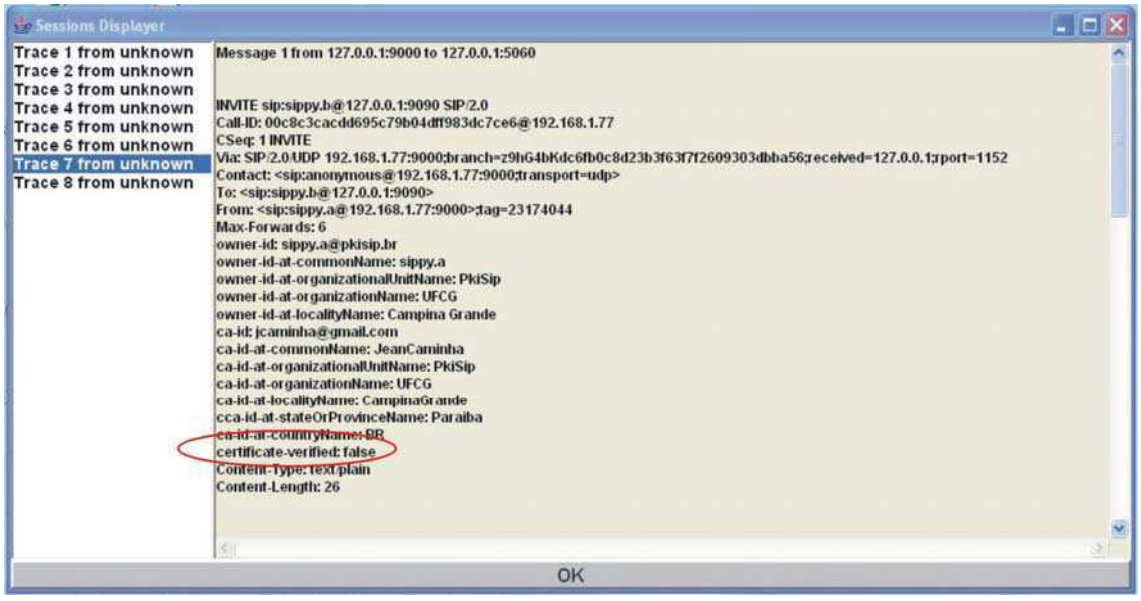

Figura 4.8 Informação da identidade não confirmada

A Figura 4.9 mostra o resultado mostrado na tela do usuário SIPPYB sobre a identidade do usuário SIPPYA.

Figura 4.9 Resultado da verificação da identidade de SIPPYA

4.6. Análise do estudo de caso

O estudo de caso mostrou a viabilidade da arquitetura proposta, tanto para o um caso de identificação positiva, quanto para uma negativa. Apesar de ser um ambiente simulado, a escolha das tecnologias utilizadas para a implementação permite a reprodução em ambiente real, sem necessidade de alterações.

Durante as pesquisas para a realização do estudo de caso, foi identificado uma certa incompatibilidade entre as assinaturas criadas pelo OpenSSL e Java. Uma assinatura criada pelo OpenSSL não pode ser lida no Java, porém, uma assinatura feita no Java era reconhecida no OpenSSL. Em pesquisas[4] em fóruns e outros sítios na internet, comprovou-se que este problema é recorrente e diversas causas foram identificadas e soluções criadas. Em nosso caso, assinamos o arquivo através do Java, com as chaves geradas pelo OpenSSL.

Destaca-se ainda que o conjunto das informações necessárias para a autenticação do usuário (certificado e arquivo assinado) tem um tamanho de aproximadamente dois bytes.

O protocolo SIP mostrou-se bastante adaptável, o que nos faz vislumbrar outras aplicações e usos com os recursos da Infra-Estrutura de Chave Pública, como a autenticação do dispositivo na rede, autenticação de aplicativos e criação de chaves de criptografia para a proteção da sessão, viabilizando transações seguras, a exemplo do TLS em navegadores de internet.

4.7. Conclusão

Neste capítulo apresentou uma implementação da arquitetura proposta, simulando uma aplicação utilizando dispositivos móveis e a infra-estrutura de chave pública. Foram abordados os casos de sucesso e insucesso, analisando os resultados obtidos.

---

[4] < http://forum.java.sun.com/thread.jspa?threadID=762354&tstart=150>

Capítulo 5 – Conclusão e futuros trabalhos

5.1. Conclusão

O objetivo deste trabalho consiste em propor uma arquitetura para a utilização da Infra-estrutura de Chave Pública (PKI) por dispositivos móveis. A arquitetura desenvolvida utiliza o do *Session Initiation Protocol* (SIP) para a troca de mensagens, utilizando os elementos necessários para a autenticação de um objeto (certificado digital e assinatura).

A arquitetura viabiliza com sucesso a autenticação de dispositivos móveis por uma Infra-estrutura de chaves pública (PKI). A utilização do *Session Initiation Protocol* (SIP) agrega valor ao sistema por ser flexível e independente de outros protocolos para tráfego de dados e também por permitir a utilização de outros recursos de segurança. A implementação compatível com Java e com sua biblioteca *Connected Limited Device Configuration* (CLDC), garante sua portabilidade para várias arquiteturas e dispositivos.

A pesquisa bibliográfica comparou a solução apresentada com outras implementações de Infra-estrutura de chaves pública (PKI), como por exemplo, conexões do tipo *ad-hoc* e cliente-servidor, verificando a legitimidade da solução apresentada.

Os resultados gerados pelo estudo de caso constataram à viabilidade da arquitetura proposta e também revelou um o cenário das dificuldades de implementação com as ferramentas escolhidas. As soluções geradas, como a inclusão de um arquivo assinado e o certificado digital no dispositivo móvel, economiza recursos do sistema e colaboram para a segurança, pois não foi necessário incluir a chave privada no dispositivo nem houve necessidade de processamento criptográfico.

A utilização do mesmo tipo de chaves de uma implementação convencional de PKI sugere uma compatibilidade e autenticação com outras arquiteturas e sistemas que utilizam este padrão, promovendo o reuso de chaves e aumentando a segurança como um todo, através da unificação de listas de revogação.

5.2. Sugestões para futuros trabalhos

Neste trabalho, foi apresentada apenas a abordagem para a autenticação e não repúdio de usuário e ações utilizando dispositivos móveis e Infra-estrutura de Chave Pública, que são somente algumas das características da segurança da informação. Outros aspectos, como a confidencialidade e disponibilidade podem ser estudados.

Como foi utilizado um ambiente simulado, uma implementação em dispositivos reais seria possível, visto que foi utilizada a linguagem de programação Java, facilmente portável. A utilização do certificado digital também poderia ser usada para autenticar o dispositivo no servidor, reforçando a segurança e ampliando o leque de aplicações.

Estudos e implementações para criação e instalação do certificado digital e do arquivo assinado no dispositivo móvel podem ser efetuados, visto a incompatibilidade

identificada no estudo de caso e também a utilização de forma mais amigável pelos usuários.

A implementação em outra linguagem, como Symbian, pode aumentar o número de dispositivos compatíveis com a solução e resolver algumas limitações da linguagem Java.

Como utiliza um Proxy SIP, pode ser possível uma implementação utilizando o repasse de informações entre uma infra-estrutura de comunicação SIP, utilizando vários servidores remotos e outros tipos de dispositivos.

A segurança na comunicação via SIP também pode ser acrescentada e estudada. Visualizamos a implementação de toda a pilha do protocolo TLS para que, além da autenticação, a conexão também possa ocorrer de forma criptografada, promovendo a confidencialidade da sessão. Os trabalhos sobre a utilização e segurança de SIP desenvolvidos pelo Laboratório de Sistemas Embarcados e Computação Pervasiva da UFCG[5] complementam os requisitos de segurança deste trabalho.

[5] <http://embedded.ufcg.edu.br/index.html>

Referências

[BING 05] BING, W.; JIE, W.; FERNANDEZA, E.; ILYASA, M.; MAGLIVERAS, S., *Secure and efficient key management in mobile ad hoc networks*, Journal of Network and Computer Applications, Elsevier, 2005, article in press.

[CAHI 03] CAHIL, V.; et al., *Using Trust for Secure Collaboration in Uncertain Environments*, Pervasive Computing, IEEE, Julho-Setembro 2003, vol. 2, no. 3, pp. 52-62.

[CAMA 02] CAMARILLO, G., *SIP Demystified*, McGraw-Hill, 2002.

[CHEU 04] CHEUNG, T.; CHANSON, S., *Design and Implementation of a PKI-based End-to-End Secure Infrastructure for Mobile E-Commerce*, World Wide Web Journal, WWWJ, 2004, vol. 4, no. 4.

[CHOU 02] CHOUDHURY, S.; BHATNAGAR, K.; HAQUE, W., *Public Key Infrastructure Implementation and Design*, Hungry Minds, 2002.

[DATT 03] DATTA, A.; HAUSWIRTH, M.; ABERER, K., *Beyond "web of trust": Enabling P2P E-commerce*, IEEE International Conference on E-Commerce (CEC'03), IEEE, 2003.

[DAVI 02] DAVIES, N.; GELLERSEN, H., *Beyond Prototypes: Challenges in Deploying Ubiquitous Systems*, Pervasive Computing, IEEE, Janeiro-Março 2002, vol. 1, no. 1, pp. 26-35.

[EPP 03] EPP, E., *Relationship Management: Secure Collaboration in a Ubiquitous Environment*, Pervasive Computing, IEEE, Abril-Junho 2003, vol. 2, no. 2, pp. 62-70.

[FORD 04] FORD, W., *Standards for Security in Open Systems*, IEEE, 1989.

[GOME 03] GOMEZ, A.; MARTINEZ, G.; CANOVAS, O., *New security services based on PKI*, Future Generation Computer Systems, Elsevier, 2003, no. 19, pp. 251-252.

[HARR 05] HARRIS, S., *CISSP All-in-One Exam Guide*, McGraw-Hill, 2005.

[HENN 06] SCHULZRINNE, H., *Session Initiation Protocol (SIP)*, <http://www.cs.columbia.edu/sip/> , acessado em Outubro de 2006.

[KAWA 02] KAWAMOTO, K.; NAKAMURA, N., *Study of Management on the Mobile Public Key Infrastructure*, IEEE, 2002.

[KOCH 03] KOCHNEV, D.; TEREKHOV, A., *Surviving Java for Mobiles*, Pervasive Computing, IEEE, Abril-Junho 2003, vol. 2, no. 2, pp. 91-95.

[KOUR 03] KOUROUTHNASSIS, P.; ROUSSOS, G., *Developing Consumer-Friendly Pervasive Retail Systems*, Pervasive Computing, IEEE, Abril-Junho 2003, vol. 2, no. 2, pp. 32-40.

[LIN 05] LIN, Y.; PANG, A., *Wireless and Mobile ALL-IP Networks*, Wiley, 2005.

[NOAK 01] NOAKES-FRY, K., *Public Key Infrastructure (PKI): Overview*, Technology Overview, Gartner, 2001.

[NOKI 06] NOKIA, *Technical specifications NOKIA: 6630*, <http://europe.nokia.com/A4142023> , acessado em Outubro de 2006.

[RIVE 77] RIVEST, R.L.; SHAMIR, A.; ADLEMAN, L., *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Massachusetts Institute of Technology, 1977.

[ROSE 02] ROSENBERG, J.; et al., *Request for Comments 3261 - SIP: Session Initiation Protocol*, ITF – Network Working Group, 2002.

[SAIL 03] SAILER, R.; GILES, J., *Pervasive Authentication Domains for Automatic Pervasive Device Authorization*, Second IEEE Annual Conference on Pervasive Computing and Communications Workshops (PERCOMW'04), IEEE, 2004.

[SCHW 06] SCHWINGENSCHLOGLA, C.; EICHLERB, S.; MULLER-RATHGEBERB, B., *Performance of PKI-based security mechanisms in mobile ad hoc networks*, International Journal of Electronic Community, vol. 60, pp. 20–24, Elsevier, 2006.

[SINN 06] SINNREICH, H.; JOHNSTON, A., *Internet communications using SIP: delivering VoIP and multimedia services with Section Initiation*, Wiley, 2006.

[STAN 02] STANFORD, V., *Pervasive Computing Goes to Work: Interfacing to the Enterprise*, Pervasive Computing, IEEE, Julho-Setembro 2002, vol. 1, no. 3, pp. 6-12.

[WOLF 05] WOLFL, T., *Public-Key-Infrastructure Based on a Peer-to-Peer Network*, 38th Hawaii International Conference on System Sciences, IEEE, 2005.

[XENI 00] XENITELLIS, S.; et al., *The Open–source PKI Book: A guide to PKIs and Open–source Implementations*, versão 2.4.6, 2000, <http://ospkibook.sourceforge.net/>, acessado em Outubro de 2006.

Anexos

Anexo A – Implementação do Servidor de Chaves

```
/*
 * RequestForwarding.java
 *
 * Created on April 16, 2003, 11:08 AM
 */

package gov.nist.sip.proxy;

import java.util.*;
import javax.sip.*;
import javax.sip.message.*;
import javax.sip.header.*;
import javax.sip.address.*;

import org.apache.log4j.Logger;

import gov.nist.sip.proxy.presenceserver.*;

/**
 *
 * @author Deruelle
 */
public class RequestForwarding {

        private static Logger logger = Logger.getLogger(RequestForwarding.class);

        protected Proxy proxy;

        /** Creates a new instance of RequestForwarding */
        public RequestForwarding(Proxy proxy) {
                this.proxy = proxy;
        }

        public void forwardRequest(Vector targetsURIList, SipProvider sipProvider,
                        Request request, ServerTransaction serverTransaction,
                        boolean statefullForwarding) {
                /*
                 * RFC 3261: 16.6. Request Forwarding For each target, the proxy
                 * forwards the request following these steps:
                 *
                 * 1. Make a copy of the received request
                 *
                 * 2. Update the Request-URI
                 *
                 * 3. Update the Max-Forwards header field
                 *
                 * 4. Optionally add a Record-route header field value
                 *
                 * 5. Optionally add additional header fields
                 *
                 * 6. Postprocess routing information
                 *
                 * 7. Determine the next-hop address, port, and transport
                 *
```

```
                 * 8. Add a Via header field value
                 *
                 * 9. Add a Content-Length header field if necessary
                 *
                 * 10. Forward the new request
                 *
                 * 11. Set timer C
                 */
                MessageFactory messageFactory = proxy.getMessageFactory();
                HeaderFactory headerFactory = proxy.getHeaderFactory();
                SipStack sipStack = proxy.getSipStack();
                AddressFactory addressFactory = proxy.getAddressFactory();
                // Get the parameters and the transport of the request URI
                URI requestURI = request.getRequestURI();
                Iterator parametersNames = null;
                /*
                 * String transport=null; System.out.debug(requestURI.toString());
                 * if(requestURI.isSipURI()){
                 * parametersNames=((SipURI)requestURI).getParameterNames();
                 * transport=((SipURI)requestURI).getTransportParam(); }
                 */
                try {

                        for (int i = 0; i < targetsURIList.size(); i++) {

                                URI targetURI = (URI) targetsURIList.elementAt(i);
                                // Copy the parameters and the transport in the new Request
URI
                                // of the cloned Request
                                /**
**************************************************************************** */
                                /** ************ 1. Make a of received request ********* */
                                /**
**************************************************************************** */
                                Request clonedRequest = (Request) request.clone();
                                if (logger.isDebugEnabled())
                                        logger
                                                        .debug("RequestForwarding,
forwardRequest() (STEP 1),"
                                                                        + " the request is
cloned");
                                /**
**************************************************************************** */
                                /** ************ 2. Update the Request-URI ********* */
                                /**
**************************************************************************** */

                                /*
                                 * The Request-URI in the copy's start line MUST be
replaced
                                 * with the URI for this target. If the URI contains any
                                 * parameters not allowed in a Request-URI, they MUST be
                                 * removed.
                                 *
                                 * This is the essence of a proxy's role. This is the
mechanism
                                 * through which a proxy routes a request toward its
                                 * destination.
                                 *
                                 * In some circumstances, the received Request-URI is
placed
                                 * into the target set without being modified. For that
target,
                                 * the replacement above is effectively a no-op.
                                 *
                                 * Note -- this should only be done if the target domain is
                                 * managed by the proxy server. Bug fix by Daniel Martinez.
                                 */
                                // All the targets URI are already canonicalized
                                if (requestURI.isSipURI()) {
                                        if (proxy.managesDomain(((SipURI)
requestURI).getHost())) {
                                                clonedRequest.setRequestURI(targetURI);
                                                if (logger.isDebugEnabled()) {
                                                        logger

        .debug("RequestForwarding, forwardRequest() (STEP 2),"
```

```
                                                                        + " The
Request-URI in the copy's start line is replaced with"
                                                                        + " the
URI for this target");
                                        }
                                } else {
                                        if (logger.isDebugEnabled()) {
                                                logger

        .debug("RequestForwarding, forwardRequest() (STEP 2),"
                                                                        + "the
proxy does not manage the domain "
                                                                        +
((SipURI) targetURI).getHost());
                                        }
                                }
                        }

                        /**
**************************************************************************** */
                        /** ************ 3. Max-Forwards ********* */
                        /**
**************************************************************************** */
                        /*
                         * If the copy contains a Max-Forwards header field, the
proxy
                         * MUST decrement its value by one (1). If the copy does
not
                         * contain a Max-Forwards header field, the proxy MUST add
one
                         * with a field value, which SHOULD be 70.
                         */

                        // RequestValidation took already care
                        // to check if the header has
                        // reached 0.
                        MaxForwardsHeader mf = (MaxForwardsHeader) clonedRequest
                                        .getHeader(MaxForwardsHeader.NAME);
                        if (mf == null) {
                                if (logger.isDebugEnabled())
                                        logger.debug("RequestForwarding,
forwardRequest() "
                                                        + " (STEP 3),
MaxForwardHeader "
                                                        + " created and added to the
cloned request");
                                mf = headerFactory.createMaxForwardsHeader(70);
                                clonedRequest.addHeader(mf);
                        } else {
                                if (logger.isDebugEnabled())
                                        logger.debug("RequestForwarding,
forwardRequest(), "
                                                        + " (STEP 3) MaxForwardHeader
"
                                                        + " decremented by one.");
                                mf.setMaxForwards(mf.getMaxForwards() - 1);
                        }

                        /**
**************************************************************************** */
                        /** ************ 4. Record-Route ********* */
                        /**
**************************************************************************** */

                        /*
                         * The URI placed in the Record-Route header field value
MUST be
                         * a SIP or SIPS URI. This URI MUST contain an lr parameter
(see
                         * Section 19.1.1). This URI MAY be different for each
                         * destination the request is forwarded to. The URI SHOULD
NOT
                         * contain the transport parameter.
                         */

                        // Only in statefull forwarding
```

```
                        // We add our proxy RecordRoute header to the top of the
list
                        // We take the first listening point.
                        String stackIPAddress = sipStack.getIPAddress();

                        ListeningPoint lp = sipProvider.getListeningPoint();
                        if (statefullForwarding) {
                                SipURI sipURI = addressFactory.createSipURI(null,
                                                stackIPAddress);
                                sipURI.setPort(lp.getPort());

                                Address address = addressFactory
                                                .createAddress(null, sipURI);
                                RecordRouteHeader recordRouteHeader = headerFactory
                                                .createRecordRouteHeader(address);

                                // lr parameter to add:
                                recordRouteHeader.setParameter("lr", null);
                                ListIterator recordRouteHeaders = clonedRequest
                                                .getHeaders(RecordRouteHeader.NAME);
                                clonedRequest.removeHeader(RecordRouteHeader.NAME);
                                Vector v = new Vector();
                                v.addElement(recordRouteHeader);
                                // add the other record route headers.
                                while (recordRouteHeaders != null
                                                && recordRouteHeaders.hasNext()) {
                                        recordRouteHeader = (RecordRouteHeader)
recordRouteHeaders
                                                        .next();
                                        v.addElement(recordRouteHeader);
                                }
                                for (int j = 0; j < v.size(); j++) {
                                        recordRouteHeader = (RecordRouteHeader)
v.elementAt(j);
                                        clonedRequest.addHeader(recordRouteHeader);
                                }

                                if (logger.isDebugEnabled())
                                        logger
                                                        .debug("RequestForwarding,
forwardRequest(), (STEP 4)"
                                                                        + " record-
route header created and added to the "
                                                                        + " cloned
request");
                        } else {
                                if (logger.isDebugEnabled())
                                        logger
                                                        .debug("RequestForwarding,
forwardRequest(), (STEP 4)"
                                                                        + " record-
route header not added to the cloned request "
                                                                        + "
(stateless)");
                        }

                        /**
*************************************************************************** */
                        /** ************ 5. Add Additional Header Fields *********
*/
                        /**
*************************************************************************** */

                        // No Additional headers to add...
                        if (logger.isDebugEnabled())
                                logger
                                                .debug("RequestForwarding,
forwardRequest(), (STEP 5)"
                                                                + " No Additional
headers to add...");

                        /**
*************************************************************************** */
                        /** ************ 6. Postprocess routing information
********* */
                        /**
*************************************************************************** */
```

```
                        /*
                         * If the copy contains a Route header field, the proxy
MUST
                         * inspect the URI in its first value. If that URI does not
                         * contain an lr parameter, the proxy MUST modify the copy
as
                         * follows: - The proxy MUST place the Request-URI into the
                         * Route header field as the last value. - The proxy MUST
then
                         * place the first Route header field value into the
Request-URI
                         * and remove that value from the Route header field.
                         */

                        // Strip first route if it is the proxy UIR adn lr
parameter
                        ListIterator routes = clonedRequest
                                        .getHeaders(RouteHeader.NAME);
                        if (routes != null && routes.hasNext()) {

                                RouteHeader routeHeader = (RouteHeader)
routes.next();
                                Address routeAddress = routeHeader.getAddress();
                                URI routeURI = routeAddress.getURI();

                                if (routeURI.isSipURI() && ((SipURI)
routeURI).hasLrParam()) {

                                        String host = ((SipURI) routeURI).getHost();

                                        if (stackIPAddress.equals(host)) {
                                                routes.remove();

                                                if (logger.isDebugEnabled())
                                                        logger
                                                                        .debug("Proxy,
forwardRequest(), (STEP 6) we removed"

        + " the first Route header field value: it contained"

        + " the proxy address and a lr parameter");
                                        } else {

                                                if (logger.isDebugEnabled())
                                                        logger

        .debug("RequestForwarding, forwardRequest(), (STEP 6)"

        + "lr parameter detected but no match...");
                                        }
                                } else if (logger.isDebugEnabled())
                                        logger
                                                        .debug("RequestForwarding,
forwardRequest(), (STEP 6)"
                                                                        + " no
Postprocess routing information to do "
                                                                        + "(the route
has no lr parameter)...");
                        } else {

                                if (logger.isDebugEnabled())
                                        logger
                                                        .debug("RequestForwarding,
forwardRequest(), (STEP 6)"
                                                                        + " no
Postprocess routing information to do (No routes detected)...");
                        }
                        /**
************************************************************************** */
                        /**
                         * 7. Determine Next-Hop Address, Port, and
                         * Transport
                         */
                        /**
************************************************************************** */
```

```
                        /*
                         * the proxy applies the procedures listed in [4] as
follows to
                         * determine where to send the request. If the proxy has
                         * reformatted the request to send to a strict-routing
element
                         * as described in step 6 above, the proxy MUST apply those
                         * procedures to the Request-URI of the request. Otherwise,
the
                         * proxy MUST apply the procedures to the first value in
the
                         * Route header field, if present, else the Request-URI.
The
                         * procedures will produce an ordered set of (address,
port,
                         * transport) tuples. Independently of which URI is being
used
                         * as input to the procedures of [4], if the Request-URI
                         * specifies a SIPS resource, the proxy MUST follow the
                         * procedures of [4] as if the input URI were a SIPS URI.
                         */
                        if (logger.isDebugEnabled())
                                logger
                                                .debug("RequestForwarding,
forwardRequest(), (STEP 7)"
                                                                + " Determine Next-Hop
Address, Port, and Transport will be done by the stack...");

                        /**
*************************************************************************** */
                        /** ************ 8. Add a Via header field value *********
*/
                        /**
*************************************************************************** */

                        /*
                         * The proxy MUST insert a Via header field value into the
copy
                         * before the existing Via header field values.
                         */


                         Iterator lps=sipStack.getListeningPoints();
                         lp=(ListeningPoint)lps.next();
                         stackIPAddress = lp.getIPAddress();


                        ViaHeader viaHeader = null;

                        if (clonedRequest.getMethod().equals(Request.CANCEL)) {
                                // Branch Id will be assigned by the stack.
                                viaHeader =
headerFactory.createViaHeader(stackIPAddress,
                                                lp.getPort(), lp.getTransport(),
null);

                                if
(clonedRequest.getMethod().equals(Request.CANCEL)) {
                                        // Cancel is hop by hop so remove all other
via headers.
                                        clonedRequest.removeHeader(ViaHeader.NAME);
                                }
                        } else {

                                viaHeader =
headerFactory.createViaHeader(stackIPAddress,
                                                lp.getPort(), lp.getTransport(),
ProxyUtilities
                                                                .generateBranchId());
                        }

                        if (viaHeader != null)
                                clonedRequest.addHeader(viaHeader);

                        if (logger.isDebugEnabled())
                                logger
```

```
                                                .debug("RequestForwarding,
forwardRequest(), (STEP 8)"
                                                        + " the proxy inserts
a Via header field value into the copy"
                                                        + " before the
existing Via header field values");

                                /*
                                 * Proxies choosing to detect loops have an additional
                                 * constraint in the value they use for construction of the
                                 * branch parameter. A proxy choosing to detect loops
SHOULD
                                 * create a branch parameter separable into two parts by
the
                                 * implementation. The first part MUST satisfy the
constraints
                                 * of Section 8.1.1.7 as described above. The second is
used to
                                 * perform loop detection and distinguish loops from
spirals.
                                 */
                                // Not yet implemented
                                if (logger.isDebugEnabled())
                                        logger
                                                .debug("RequestForwarding,
forwardRequest(), (STEP 8)"
                                                        + " Loop detection not
implemented");

                                /**
*************************************************************************** */
                                /**
                                 *  9. Add a Content-Length header field if necessary
                                 *
                                 */
                                /**
*************************************************************************** */

                                try {
                                        ContentTypeHeader contentTypeHeader =
(ContentTypeHeader) clonedRequest
                                                        .getHeader(ContentTypeHeader.NAME);
                                        if (contentTypeHeader == null) {
                                                if (logger.isDebugEnabled())
                                                        logger.debug("RequestForwarding,
forwardRequest(),"
                                                                + " no Content-Type
header,"
                                                                + " we don't stripe
any parameters!!!");
                                        } else
                                                contentTypeHeader.removeParameter("msgr");

                                } catch (Exception e) {
                                        logger.debug("RequestForwarding, forwardRequest(),
Stripe"
                                                        + " Parameter failed!!!!");
                                }

                                /**
*************************************************************************** */
                                /** ********* 10. Forward Request ********* */
                                /**
*************************************************************************** */

                                if (statefullForwarding) {
                                        forwardRequestStatefully(sipProvider,
clonedRequest,
                                                        request, serverTransaction);
                                } else {
                                        forwardRequestStatelessly(sipProvider,
clonedRequest,
                                                        request, serverTransaction);
                                }

                                /**
*************************************************************************** */
```

```
                        /** ******** 11. Set timer C ********* */
                        /**
************************************************************************ */

                        // Not Implemented....
                }
        } catch (Exception ex) {
                try {
                        logger.error("RequestForwarding, forwardRequest(), "
                                        + " internal error, " + "exception raised:",
ex);

                        // This is an internal error:
                        // Let's return a 500 SERVER_INTERNAL_ERROR
                        Response response = messageFactory.createResponse(
                                        Response.SERVER_INTERNAL_ERROR, request);
                        if (serverTransaction != null)
                                serverTransaction.sendResponse(response);
                        else
                                sipProvider.sendResponse(response);

                        if (logger.isDebugEnabled())
                                logger.debug("RequestForwarding, forwardRequest(),
"
                                                + " 500 SERVER_INTERNAL_ERROR
replied:\n"
                                                + response.toString());
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
    }

    /**
     * Forward the request statefully.
     *
     * @param sipProvider --
     *            sip provider to forward request.
     * @param clonedRequest --
     *            cloned request to forward.
     * @param originalRequest --
     *            incoming request
     * @param serverTransaction --
     *            server transaction used to fwd the request.
     */

    private void forwardRequestStatefully(SipProvider sipProvider,
                    Request clonedRequest, Request originalRequest,
                    ServerTransaction serverTransaction) {
            MessageFactory messageFactory = proxy.getMessageFactory();
            if (logger.isDebugEnabled()) {
                    logger.debug("serverTransaction =  " + serverTransaction);
                    if (serverTransaction != null)
                            logger.debug("dialog =  " + serverTransaction.getDialog());
            }

            try {
                    /*
                     * A stateful proxy MUST create a new client transaction for this
                     * request as described in Section 17.1 and instructs the
                     * transaction to send the request using the address, port and
                     * transport determined in step 7
                     */
                    if (serverTransaction == null
                                    &&
!clonedRequest.getMethod().equals(Request.MESSAGE)) {
                            // dont create a server transaction for MESSAGE -
                            // just forward the request statefully through a new
                            // client transaction.
                            if (logger.isDebugEnabled())
                                    logger
                                                    .debug("RequestForwarding,
forwardRequestStatefully(),"
                                                                    + " the cloned request
does not have a server transaction,"
                                                                    + " so we drop the
request!");
```

```
                        return;
                }

                if (originalRequest.getMethod().equals(Request.CANCEL)) {
                        // 487 Request Terminated to reply:
                        Response response = messageFactory.createResponse(
                                        Response.REQUEST_TERMINATED,
originalRequest);
                        CSeqHeader cSeqHeader = (CSeqHeader) response
                                        .getHeader(CSeqHeader.NAME);
                        cSeqHeader.setMethod("INVITE");
                        serverTransaction.sendResponse(response);

                        if (logger.isDebugEnabled())
                                logger
                                                .debug("Proxy, processRequest(), 487
Request Terminated"
                                                                + " replied to the
CANCEL:\n"
                                                                +
response.toString());
                }

                if (logger.isDebugEnabled())
                        logger.debug("RequestForwarding,
forwardRequestStatefully(),"
                                        + " the dialog state is null, so we have to"
                                        + " forward the request using"
                                        + " a new clientTransaction");
                ClientTransaction clientTransaction = sipProvider
                                .getNewClientTransaction(clonedRequest);
                clientTransaction.sendRequest();

                if (logger.isDebugEnabled())
                        logger.debug("RequestForwarding,
forwardRequestStatefully(),"
                                        + ", cloned request forwarded statefully:\n
"
                                        + clonedRequest);

                TransactionsMapping transactionsMapping = (TransactionsMapping)
serverTransaction
                                .getApplicationData();
                if (transactionsMapping != null)
                        transactionsMapping.addMapping(serverTransaction,
                                        clientTransaction);
                else {
                        transactionsMapping = new TransactionsMapping();
                        transactionsMapping.addMapping(serverTransaction,
                                        clientTransaction);
                        serverTransaction.setApplicationData(transactionsMapping);
                        clientTransaction.setApplicationData(transactionsMapping);

                }

        } catch (Exception ex) {
                try {
                        if (logger.isDebugEnabled()) {
                                logger
                                                .debug(
                                                                "RequestForwarding,
forwardRequestStatefully(), "
                                                                                + "
internal error, "
                                                                                +
"exception raised:", ex);
                                ex.printStackTrace();
                                System.exit(0);
                        }
                        // This is an internal error:
                        // Let's return a 500 SERVER_INTERNAL_ERROR
                        Response response = messageFactory.createResponse(
                                        Response.SERVER_INTERNAL_ERROR,
originalRequest);
                        if (serverTransaction != null)
                                serverTransaction.sendResponse(response);
                        else
```

```
                                        sipProvider.sendResponse(response);

                                if (logger.isDebugEnabled())
                                        logger
                                                        .debug("RequestForwarding,
forwardRequestStatefully(), "
                                                                        + " 500
SERVER_INTERNAL_ERROR replied:\n"
                                                                        +
response.toString());
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                }
        }

        private void forwardRequestStatelessly(SipProvider sipProvider,
                        Request clonedRequest, Request originalRequest,
                        ServerTransaction serverTransaction) {

                MessageFactory messageFactory = proxy.getMessageFactory();
                HeaderFactory headerFactory = proxy.getHeaderFactory();
                AddressFactory addressFactory = proxy.getAddressFactory();
                SipStack sipStack = proxy.getSipStack();

                try {

                        // We forward statelessly:
                        // It means the Request does not create dialogs...
                        sipProvider.sendRequest(clonedRequest);
                        if (logger.isDebugEnabled())
                                logger.debug("RequestForwarding,
forwardRequestStatelessly(), "
                                                + " cloned request forwarded statelessly:\n"
                                                + clonedRequest.toString());
                } catch (Exception ex) {
                        try {
                                logger.error("RequestForwarding,
forwardRequestStatelessly(), "
                                                + " internal error, " + "exception raised:",
ex);

                                // This is an internal error:
                                // Let's return a 500 SERVER_INTERNAL_ERROR
                                Response response = messageFactory.createResponse(
                                                Response.SERVER_INTERNAL_ERROR,
originalRequest);
                                if (serverTransaction != null)
                                        serverTransaction.sendResponse(response);
                                else
                                        sipProvider.sendResponse(response);

                                if (logger.isDebugEnabled())
                                        logger
                                                        .debug("RequestForwarding,
forwardRequestStatelessly(), "
                                                                        + " 500
SERVER_INTERNAL_ERROR replied:\n"
                                                                        +
response.toString());
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                }

                private boolean validateCertificate(Request request) throws
InvalidKeyExeption, NoSuchAlgorithmException, SignatureException,
InvalidKeySpecException{

                        ExtensionHeaderImpl ownerID =
(ExtensionHeaderImpl)request.getHeader("owner-id");
                        ExtensionHeaderImpl ownerPublicKey =
(ExtensionHeaderImpl)request.getHeader("owner-public-Key");
                        ExtensionHeaderImpl ownerEncrypted =
(ExtensionHeaderImpl)request.getHeader("owner-encrypted");

                        byte[] data = ownerID = ownerID.getValue().getBytes();
```

```
                        byte[] sigData = ownerEncrypted.getValue().getBytes();
                        PublicKey publicKey  = getPublicKey(ownerPublicKey.getValue());
                        return this.verify(publicKey,data,sigData);
                }


                private PublicKey getPublicKey(String value) throws IOException,
NoSuchAlgorithmException, InvalidKeySpecException {

                        ByteArrayInputStream in = new
ByteArrayInputStream(value.getBytes());
                        byte[] data = new byte[in.available()];
                        in.read(data);
                        in.close();

                        X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec(data);
                        KeyFactory factory = KeyFactory.getInstance("RSA");
                        PublicKey pubKey = factory.generatePublic(pubKeySpec);

                        return pubKey;
        }

}
```

Anexo B – Implementação do Dispositivo Móvel SippyA

```
/**
 * @(#)BaseUAC.java   1.2 05/10/18
 *
 * Copyright © 2000-2006 Sun Microsystems, Inc. All rights reserved.
 * PROPRIETARY/CONFIDENTIAL
 * Use is subject to license terms
 *
 * BaseUAC.java
 *
 * Created on October 10, 2005, 12:10 PM
 */

package example.sip;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import javax.microedition.io.Connector;
import javax.microedition.io.ServerSocketConnection;
import javax.microedition.io.SocketConnection;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.sip.SipClientConnection;
import javax.microedition.sip.SipClientConnectionListener;
import javax.microedition.sip.SipConnectionNotifier;
import javax.microedition.sip.SipDialog;
import javax.microedition.sip.SipException;
import javax.microedition.sip.SipServerConnection;
import javax.microedition.sip.SipServerConnectionListener;

/**
 * Application demonstrates sip clients talking via sip server and registrar.
 * Both clients register to sip registrar. Send connection informations
 * in INVITE message to each other and start to communicate via sockets.
 *
 * Application demonstrates REGISTER, INVITE, BYE requests and
 * OK, RINGING responses.
 *
 * @version 1.2
 */
public abstract class BaseUAC extends MIDlet implements CommandListener,
                                          SipClientConnectionListener,
                                          SipServerConnectionListener {

    /** client states */
//    private static final int DISCONNECTED = 0;
    private static final int REGISTERING = 1;
    private static final int REGISTERED = 2;
    private static final int INVITING = 3;
    private static final int TALKING = 4;
    private static final int RINGING = 5;
    private static final int BYE = 6;

    /** user name if the client */
    public String myName = "a";
```

```
/** display name of the client */
public String myDisplayName = "A";

/** socket used by this client */
public int mySocket = 1111;

/** sip port the client is listening on */
public int mySipPort = 9000;

/** user name of second client */
public String friendName = "B";

/** socket used by second client */
public int friendSocket = 2222;

/** sip port of second client
 * we need this because running both
 * clients on the same machine. That isn't
 * typical use case.
 */
public int friendSipPort = 9090;

/** second client's domain */
public String friendDomain = "localhost";

/** forms used in ui */
protected Form proxyFrm = null;
protected Form registerFrm = null;
protected Form waitScreen = null;
protected Form inviteFrm = null;
protected Form talkFrm = null;
protected Form sendFrm = null;
protected Form failFrm = null;
protected Form ringingFrm = null;
protected Form byeFrm = null;

private Form backupForm = null;

private Gauge progressGauge = null;

/** commands used in ui */
private Command exitCmd = new Command("Exit", Command.EXIT, 1);
private Command registerCmd  = new Command("Register", Command.OK, 1);
private Command backCmd = new Command("Back", Command.BACK, 1);
private Command nextCmd = new Command("Next", Command.SCREEN, 1);
private Command inviteCmd = new Command("Invite", Command.OK, 1);
private Command sendCmd = new Command("Send", Command.OK, 1);
private Command okCmd = new Command("Ok", Command.OK, 1);
private Command failedCmd = new Command("Failed", Command.BACK, 1);
private Command denyCmd = new Command("Deny", Command.EXIT, 1);
private Command answerCmd = new Command("Answer", Command.OK, 1);
private Command byeCmd = new Command("Bye", Command.BACK, 1);

private Displayable currentDisplay, backDisplay;
private Display display;

/** receive thread runs forewer */
private Thread receiveThread = null;

/** gauge  */
private Thread gaugeThread = null;
private boolean progressGaugeFinished = true;

/** socket streams */
private InputStream socketIStream;
private OutputStream socketOStream;

/** server socket */
private ServerSocketConnection serverSocket = null;

/** client socket */
private SocketConnection sc = null;

/** sip variables */
private String proxyAddress = "";
private SipConnectionNotifier scn;
```

```
private String failMessage;
private SipClientConnection scc;
private SipServerConnection ssc;
private SipDialog dialog;
private String clientSockParams;
private Sender sender;

/** appliaction status */
private Status uaStatus = new Status();

/** how long the gauge should run (simplified)*/
private int finishGauge;

public void start() {
    display = Display.getDisplay(this);
    if (proxyAddress==null || proxyAddress.length() == 0) {
        setDisplay(getProxyFrm());
    } else {
        setDisplay(getRegisterForm());
    }
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional) {
    System.out.println("Closing app ...");
    tearDown();
    notifyDestroyed();
}

public void setProxyAddress(String address) {
    proxyAddress = address;
}

public void setSocket(int socket) {
    mySocket = socket;
}

/** Setup all params of the client.
 *
 * @param name username of the client.
 * @param displayName the display name of the user.
 * @param port sip listener port
 */
public void setUserIdentity(String name, String displayName,
                            int port) {
    myName = name;
    myDisplayName = displayName;
    mySipPort = port;
}

/** Setup all params of 2nd client.
 *
 * @param name the username of 2nd client
 * @param domain the domain of 2nd client
 * @param port the listener sip port of 2nd client
 */
public void setFriendIdentity(String name, String domain, int port) {
    friendName = name;
    friendDomain = domain;
    friendSipPort = port;
}

/**
 * Display another screen
 */
private void setDisplay(Displayable d) {
    if (currentDisplay!=waitScreen)
        backDisplay = currentDisplay;
    display.setCurrent(d);
    currentDisplay = d;
}

private Form getProxyFrm() {
    if (proxyFrm==null) {
       if (proxyAddress.length() == 0) {
```

```
                proxyAddress = System.getProperty("microedition.hostname");
                if (proxyAddress == null) {
                        proxyAddress = "";
                }
            }
            proxyFrm = new Form("Proxy setup",
                    new Item[] {new TextField("Proxy host:", proxyAddress, 20,
TextField.ANY)});
            proxyFrm.addCommand(nextCmd);
            proxyFrm.addCommand(exitCmd);
            proxyFrm.setCommandListener(this);
        }
        return proxyFrm;
    }

    private Form getRingingFrm(String message) {
        if (ringingFrm==null) {
            ringingFrm = new Form("Ringing ...");
            ringingFrm.append(new StringItem("Message: sippya (identidade não
verificada) deseja inciar uma sessão ", message));
            ringingFrm.addCommand(denyCmd);
            ringingFrm.addCommand(answerCmd);
            ringingFrm.setCommandListener(this);
        }
        StringItem si = (StringItem)ringingFrm.get(0);
        si.setText(message);
        return ringingFrm;
    }

    private Form getRegisterForm() {
        if (registerFrm==null) {
            registerFrm = new Form("Welcome",
            new Item[] {new StringItem(null,
                        "You need to register to the registrar server.\n" +
                        "Please press register and wait for response.")});

            registerFrm.addCommand(exitCmd);
            registerFrm.addCommand(registerCmd);
            registerFrm.setCommandListener(this);
        }
        return registerFrm;
    }

    private Form getWaitScreen(String title, int finishAfter, Form bForm) {
        if (waitScreen==null) {
            progressGauge = new Gauge(title, false, 10, 0);
            progressGauge.setLayout( Item.LAYOUT_CENTER |
                                     Item.LAYOUT_EXPAND |
                                     Item.LAYOUT_VCENTER);
            waitScreen = new Form("");
            waitScreen.append(progressGauge);
            waitScreen.addCommand(backCmd);
            waitScreen.setCommandListener(this);
        }
        finishGauge = (finishAfter==0) ? Integer.MAX_VALUE : finishAfter;
        backupForm = bForm;
        progressGauge.setLabel(title);
        progressGauge.setValue(0);
        return waitScreen;
    }

    private Form getInviteForm() {
        if (inviteFrm==null) {
            inviteFrm = new Form("Invite",
                        new Item[] {new StringItem(null,
                                "Invite your sip friend " +
                                friendName + " to talk.")});
            inviteFrm.addCommand(exitCmd);
            inviteFrm.addCommand(inviteCmd);
            inviteFrm.setCommandListener(this);

        }
        return inviteFrm;
    }

    private Form getTalkForm() {
        if (talkFrm==null) {
```

```
            talkFrm = new Form("Talking", new Item[] {});
            talkFrm.addCommand(sendCmd);
            talkFrm.addCommand(byeCmd);
            talkFrm.setCommandListener(this);
        }
        return talkFrm;
    }

    private Form getSendForm() {
        if (sendFrm==null) {
            sendFrm = new Form("Send",
                        new Item[] {new TextField("Enter message:", "", 255,
TextField.ANY)});
            sendFrm.addCommand(okCmd);
            sendFrm.addCommand(byeCmd);
            sendFrm.setCommandListener(this);
        }
        TextField tfield = (TextField)sendFrm.get(0);
        if (tfield!=null) {
            tfield.setString("");
        }
        return sendFrm;
    }

    private Form getFailFrm(String msg) {
        if (failFrm==null) {
            failFrm = new Form("Error");
            failFrm.addCommand(exitCmd);
            failFrm.setCommandListener(this);
            failFrm.append(new StringItem("", ""));
        }
        StringItem si = (StringItem)failFrm.get(0);
        si.setText(msg);
        return failFrm;
    }

    private Form getByeFrm() {
        if (byeFrm==null) {
            byeFrm = new Form("BYE");
            byeFrm.append("Received BYE from 2nd terminal.\nTerminating GoSIP session
...");
            byeFrm.addCommand(okCmd);
            byeFrm.setCommandListener(this);
        }
        return byeFrm;
    }

    public void commandAction(Command command, Displayable displayable) {
        if (command==exitCmd && displayable==registerFrm) {
            destroyApp(false);
        } else if (command==exitCmd && displayable==failFrm) {
            destroyApp(false);
        } else if (command==registerCmd && displayable==registerFrm) {
            setDisplay(getWaitScreen("Registration pending ...", 0, null));
            Thread t = listen(this);
            register(this, t);
        } else if (command==backCmd && displayable==waitScreen) {
            setDisplay(backDisplay);
        } else if (command==nextCmd &&
                   displayable==waitScreen &&
                   backDisplay==registerFrm ) {
            setDisplay(getInviteForm());
        } else if (command==nextCmd &&
                   displayable==waitScreen &&
                   backDisplay==inviteFrm) {
            setDisplay(getTalkForm());
        } else if (command==backCmd &&
                   displayable==waitScreen &&
                   backDisplay==registerFrm) {
            setDisplay(getRegisterForm());
        } else if (command==backCmd &&
                   displayable==waitScreen &&
                   backDisplay==inviteFrm) {
            setDisplay(getInviteForm());
        } else if (command==failedCmd &&
                     displayable==waitScreen &&
                     backDisplay==registerFrm) {
```

```
                setDisplay(getFailFrm("Failed to register:\n Cause: " + failMessage));
        } else if (command==failedCmd &&
                    displayable==waitScreen &&
                    backDisplay==inviteFrm) {
                setDisplay(getFailFrm("Failed to invite:\n Cause: " + failMessage));
        } else if (command==byeCmd &&
                   displayable==waitScreen) {
            setDisplay(getInviteForm());
        } else if (command==denyCmd && displayable==ringingFrm) {
            sendCancel();
        } else if (command==exitCmd && displayable==proxyFrm) {
            destroyApp(false);
        } else if (command==nextCmd && displayable==proxyFrm) {
            TextField tfield = (TextField)proxyFrm.get(0);
           String proxy = tfield.getString();
           if (proxy.length() == 0 || isIPAddress(proxy)) {
              if (proxyFrm.size() == 1) {
                    proxyFrm.append(new StringItem(null,
                    "Proxy name can't be empty or plain ip addres. Use valid
hostname."));
                    setDisplay(getProxyFrm());
              }
           } else {
              if (proxyFrm.size() > 1) {
                    proxyFrm.delete(1);
              }
               setProxyAddress(proxy);
               setDisplay(getRegisterForm());
           }
        } else if (command==answerCmd && displayable==ringingFrm) {
            setDisplay(getWaitScreen("Accepting ...", 0, null));
            sendAccepted();
            try {Thread.currentThread().sleep(1500);} catch (Exception e) {}
            if (uaStatus.getStatus()==RINGING) {
                 openClientConnection(clientSockParams);
                 uaStatus.setStatus(TALKING);
                 stopGauge();
                 commandAction(nextCmd, currentDisplay);
             }
        } else if (command==inviteCmd && displayable==inviteFrm) {
            setDisplay(getWaitScreen("Invite pending ...", 0, null));
            invite(this);
        } else if (command==byeCmd && displayable==talkFrm) {
            if (uaStatus.getStatus()==TALKING) {
                setDisplay(getWaitScreen("Bye ...", 10, getInviteForm()));
                sendBye();
            } else if (uaStatus.getStatus()==REGISTERED) {
                //do nothing
                setDisplay(getInviteForm());
            }
        } else if (command==sendCmd && displayable==talkFrm) {
            setDisplay(getSendForm());
        } else if (command==nextCmd &&
                   displayable==waitScreen &&
                   backDisplay==ringingFrm) {
            setDisplay(getTalkForm());
        } else if (command==okCmd && displayable==byeFrm) {
            setDisplay(getInviteForm());
        } else if (command==okCmd && displayable==sendFrm) {
            TextField txtField = (TextField)sendFrm.get(0);
            if (txtField != null && txtField.getString().length() > 0) {
                send(txtField.getString());
            }
            setDisplay(getTalkForm());
        } else if (command==backCmd && displayable==sendFrm) {
            setDisplay(backDisplay);
        } else if (command==exitCmd && displayable==inviteFrm) {
            destroyApp(false);
        }
    }


    private Thread listen(final SipServerConnectionListener listener) {
        Thread t = new Thread() {
            public void run() {
                try {
                    if(scn != null) {
```

```
                        scn.close();
                    }
                    scn = (SipConnectionNotifier) Connector.open("sip:" + mySipPort);
                    scn.setListener(listener);
                    try {Thread.currentThread().sleep((1000));} catch (Exception e) {}

                } catch(IOException ex) {
                    ex.printStackTrace();
                }
            }
        };
        t.start();
        return t;
    }

    private void register(final SipClientConnectionListener listener,
                          final Thread waitFor) {
        Thread t = new Thread() {
            public void run() {
                runGauge();
                try {
                      try {
                          if (waitFor!=null) {
                              waitFor.join();
                          } else {

                          }
                      } catch (InterruptedException ie) {}

                      scc = (SipClientConnection)
                            Connector.open("sip:" + proxyAddress +
":5060;transport=udp");
                      scc.setListener(listener);
                      scc.initRequest("REGISTER", scn);
                      String adr = myDisplayName + " <sip:" + myName + "@" +
                                   scn.getLocalAddress() + ":" +
                                   scn.getLocalPort() + ">";
                      scc.setHeader("To", adr);
                      scc.setHeader("From", adr);
                      scc.setHeader("Content-Length", "0");
                      scc.setHeader("Max-Forwards", "6");
                      uaStatus.setStatus(REGISTERING);
                      scc.send();
                      uaStatus.waitUntilChanged();
                      progressGaugeFinished = true;

                } catch (Exception e) {
                    e.printStackTrace();
                    failMessage = e.getMessage();
                    commandAction(failedCmd, currentDisplay);
                    return;
                }
            }
        };
        t.start();
    }

    private void invite(final SipClientConnectionListener listener) {
        Thread t = new Thread() {
            public void run() {
                runGauge();
                try {
                      String host = scn.getLocalAddress();
                      String adr = "sip:" + myName + "@" +
                                   host + ":" +
                                   scn.getLocalPort();

                      String toAdr = "sip:" + friendName + "@" +
                                     friendDomain + ":" +
                                     friendSipPort;

                      scc = (SipClientConnection)Connector.open(toAdr);
                      scc.setListener(listener);

                      scc.initRequest("INVITE", scn);

                      String message = "socket://" + host + ":" + mySocket;
```

```
                    scc.setHeader("To", toAdr);
                    scc.setHeader("From", adr);
                    scc.setHeader("Content-Type", "text/plain");
                    scc.setHeader("Content-Length",
Integer.toString(message.length()));
                    scc.setHeader("owner-id", "sippy.a@pkisip.br");
                    scc.setHeader("owner-id-at-commonName", "sippy.a");
                    scc.setHeader("owner-id-at-organizationalUnitName", "PkiSip");
                    scc.setHeader("owner-id-at-organizationName", "UFCG");
                    scc.setHeader("owner-id-localityName", "Campina Grande");
                    scc.setHeader("ca-id", "jcaminha@gmail.com");
                    scc.setHeader("ca-id-at-commonName", "JeanCaminha");
                    scc.setHeader("ca-id-at-organizationalUnitName", "PkiSip");
                    scc.setHeader("ca-id-at-organizationName", "UFCG");
                    scc.setHeader("ca-id-at-localityName", "CampinaGrande");
                    scc.setHeader("ca-id-at-coutryName", "BR");
                    scc.setHeader("owner-public-key",
"MIIEnjCCA4agAwIBAgIJAIohbW37CLeOMAOGCSqGSIb3DQEBBQUAMIGQMQswCQYD");
                    scc.setHeader("owner-encrypted",
"VQQGEwjCUjEQMA4GA1UECBMHUGFyYWLiYTEWMBxMNQ2FtcGLuYUdyYW5k");


                    OutputStream os = scc.openContentOutputStream();
                    os.write(message.getBytes());
                    uaStatus.setStatus(INVITING);
                    os.close(); // close and send
                    uaStatus.waitUntilChanged();

                    //Thread.currentThread().sleep(4000);
                    progressGaugeFinished = true;

                } catch (Exception e) {
                    e.printStackTrace();
                    failMessage = e.getMessage();
                    stopGauge();
                    commandAction(failedCmd, currentDisplay);
                    return;
                }
            }
        };
        t.start();
    }

    private synchronized void tearDown() {
        try {
            if (socketIStream!=null)
                socketIStream.close();
            if (socketOStream!=null)
                socketOStream.close();
            if (sender!=null)
                sender.stop();
            if (scc!=null)
                scc.close();
            if (ssc!=null)
                ssc.close();
            if (scn!=null) {
                scn.close();
            }
            if (sc!=null)
                sc.close();
            if (serverSocket!=null)
                serverSocket.close();
        } catch (IOException e) {}
    }

    private void openServerConnection(int socket) {
        Thread t = new Thread() {
            public void run() {
                try {
                    serverSocket = (ServerSocketConnection)
                                    Connector.open("socket://:" + mySocket);
                    System.out.println("waiting for connection socket://:" + mySocket);
                    SocketConnection sc = (SocketConnection)
serverSocket.acceptAndOpen();
                    System.out.println("connection accepted from: " + sc.getAddress());
                    socketIStream = sc.openInputStream();
```

```
                    socketOStream = sc.openOutputStream();
                    sender = new Sender(socketOStream);
                    while (true) {
                        StringBuffer sb = new StringBuffer();
                        int c = 0;
                        while (((c = socketIStream.read()) != '\n') && (c != -1)) {
                            sb.append((char) c);
                        }
                        if (c == -1) {
                            break;
                        }
                        getTalkForm().append(new StringItem(friendName + ":",
sb.toString()));
                    }//while
                } catch (IOException ioe) {
                    ioe.printStackTrace();
                    failMessage = "Cannot open server socket";
                } finally {
                    tearDown();
                }
            }
        };
        t.start();
    }

    private void openClientConnection(final String params) {
        receiveThread = new Thread() {
            public void run() {
                try {
                    //open socket
                    sc = (SocketConnection) Connector.open(params);
                    System.out.println("Client opened connection " + params);
                    socketIStream = sc.openInputStream();
                    socketOStream = sc.openOutputStream();
                    sender = new Sender(socketOStream);

                    //Loop forever, receiving data
                    while (true) {
                        StringBuffer sb = new StringBuffer();
                        int c = 0;
                        while (((c = socketIStream.read()) != '\n') && (c != -1)) {
                            sb.append((char) c);
                        }
                        if (c == -1) {
                            break;
                        }
                        // Display message to user
                        getTalkForm().append(new StringItem(friendName, sb.toString()));
                    }

                } catch (IOException ex) {
                    ex.printStackTrace();
                } finally {
                    tearDown();
                }
            }
        };
        receiveThread.start();
    }

    private void send(final String message) {
        getTalkForm().append(new StringItem(myName + ":", message));
        if (sender!=null)
            sender.send(message);
    }

    private void sendBye() {
        Thread t = new Thread() {
            public void run() {
                runGauge();
                if(dialog != null &&
                   dialog.getState() == SipDialog.CONFIRMED) {
                    try {
                        SipClientConnection sc = dialog.getNewClientConnection("BYE");
                        sc.send();
                        System.out.println("Sending BYE ...");
                        uaStatus.setStatus(BYE);
```

```
                } catch (Exception e) {
                    e.printStackTrace();
                }
            } else {
                System.out.println("Dialog isn't initialized. Cannot send BYE.");
            }
        }
    };
    t.start();
}

private void runGauge() {
    if (gaugeThread==null || !gaugeThread.isAlive()) {
        gaugeThread = new Thread() {
            public void run() {
                progressGaugeFinished = false;
                progressGauge.setValue(0);
                boolean up = true;
                int i = 0;
                int c = 0;
                while(!progressGaugeFinished &&
                      c<finishGauge) {
                    progressGauge.setValue((up) ? i++ : i--);
                    up = (i==0 ||
                          i==progressGauge.getMaxValue()) ?
                        up = !up : up;
                    try {
                        Thread.currentThread().sleep(100);
                    } catch (InterruptedException e) {}
                    c++;
                }//while
                if (backupForm!=null) setDisplay(backupForm);
                progressGaugeFinished = true;
            }
        };
        gaugeThread.start();
    }
}

private void stopGauge() {
    progressGaugeFinished = true;
}

public void notifyResponse(SipClientConnection sipClientConnection) {
    try {
       boolean ok = scc.receive(1000);
       if (!ok) {
           System.out.println("Response not received");
           return;
       }
       int code = scc.getStatusCode();
       System.out.println("Received status code: " + code);

       if (uaStatus.getStatus()==REGISTERING) {
           if (code==200) {
               System.out.println("Received OK after REGISTER");
               uaStatus.setStatus(REGISTERED);
               stopGauge();
               commandAction(nextCmd, currentDisplay);
           } else {
               stopGauge();
               failMessage = "Unknown response: " + code;
               commandAction(failedCmd, currentDisplay);
           }
       } else if (uaStatus.getStatus()==INVITING) {
           if (code>=100 && code<200) {
                System.out.println("Provisioning response: " + code);
           } else if (code==200) {
                System.out.println("Received OK after INVITE");
                scc.initAck(); // initialize and send ACK
                scc.send();
                dialog = scc.getDialog();
                uaStatus.setStatus(TALKING);
                openServerConnection(mySocket);
                stopGauge();
                commandAction(nextCmd, currentDisplay);
```

```
            } else if (code==486) { //BUSY HERE
                stopGauge();
                failMessage = "2nd client is busy.";
                commandAction(failedCmd, currentDisplay);
            } else {
                stopGauge();
                failMessage = "Unknown response: " + code;
                commandAction(failedCmd, currentDisplay);
            }
        } else if (uaStatus.getStatus()==BYE) {
            if (code==200) {
                System.out.println("Received OK after BYE");
                uaStatus.setStatus(REGISTERED);
                ssc.close();
                stopGauge();
                commandAction(byeCmd, currentDisplay);
            }
        }

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

public void notifyRequest(SipConnectionNotifier sipConnectionNotifier) {
    try {
        ssc = sipConnectionNotifier.acceptAndOpen(); // blocking
        System.out.println("Received request: " + ssc.getMethod().toString());
        if(ssc.getMethod().equals("INVITE")) {
            String contentType = ssc.getHeader("Content-Type");
            String contentLength = ssc.getHeader("Content-Length");
            int length = Integer.parseInt(contentLength);
            if(contentType.equals("text/plain")) {
                InputStream is = ssc.openContentInputStream();
                byte content[] = new byte[length];
                is.read(content);
                clientSockParams = new String(content);
                //parse socket connection params
                uaStatus.setStatus(RINGING);
                setDisplay(getRingingFrm(clientSockParams));
                ssc.initResponse(180); //RINGING
                ssc.send();
                dialog = ssc.getDialog();
            }
        } else if (ssc.getMethod().equals("ACK")) {
            if (uaStatus.getStatus()==RINGING) {
                openClientConnection(clientSockParams);
                uaStatus.setStatus(TALKING);
                stopGauge();
                commandAction(nextCmd, currentDisplay);
            }
        } else if (ssc.getMethod().equals("BYE")) {
            if (dialog.isSameDialog(ssc)) {
                setDisplay(getByeFrm());
                if (uaStatus.getStatus()==TALKING) {
                    ssc.initResponse(200);
                    ssc.send();
                    getTalkForm().setTitle("Client disconnected !");
                    try {
                        if (serverSocket != null) {
                            serverSocket.close();
                        } else if (sc != null) {
                            sc.close();
                        }
                    } catch (Exception e) {};
                    uaStatus.setStatus(REGISTERED);
                }
            } else {
                System.out.println("Not a same dialog");
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void sendCancel() {
```

```
    Thread t = new Thread() {
        public void run() {
            try {
                ssc.initResponse(486);//BUSY HERE
                ssc.send();

            } catch (Exception e) {
                System.out.println("Exception when sending cancel");
                e.printStackTrace();
            }
        }
    };
    t.start();
}

private void sendAccepted() {
    Thread t = new Thread() {
        public void run() {
            try {
                ssc.initResponse(200);//OK
                ssc.send();
                // save Dialog
                dialog = ssc.getDialog();
                ssc.close();

            } catch (Exception e) {
                System.out.println("Exception while sending OK");
                e.printStackTrace();
            }
        }
    };
    t.start();
}

private boolean isIPAddress(String address) {
    String addrex = address + ".";
    char c;
    int digcount = 0;
    int numcount = 0;

    for(int i = 0; i<addrex.length(); i++) {
        c = addrex.charAt(i);
        if (c == '.') {
            digcount = 0;
            numcount++;
            if (numcount > 4) {
                return false;
            }
        } else {
            if ('0' <= c && c <= '9') {
                digcount++;
                if (digcount > 3) {
                   return false;
                }
            } else {
              return false;
            }
        }
    }
    return (numcount == 4);
}


/**
 * Help to keep status of application.
 *
 */
class Status {
    private boolean changed = false;
    private int status = 0;

    public int getStatus() {
        return status;
    }

    public synchronized int waitUntilChanged() {
        while(!changed) {
```

```
                try {
                    wait();
                } catch (InterruptedException e) {}
            }
            changed = false;
            return status;
        }

        public synchronized void setStatus(int status) {
            this.status = status;
            changed = true;
            notify();
        }
    }

}
```