UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

SOFTWARE DE COMUNICAÇÃO VOIP COM CANAL SEGURO NA PLATAFORMA ANDROID

ANDRÉ LUIZ LEHMANN

BLUMENAU
2010

2010/2

ANDRÉ LUIZ LEHMANN

SOFTWARE DE COMUNICAÇÃO VOIP COM CANAL SEGURO NA PLATAFORMA ANDROID

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

Prof. Paulo Fernando da Silva - Orientador

BLUMENAU
2010

2010/2

SOFTWARE DE COMUNICAÇÃO VOIP COM CANAL SEGURO NA PLATAFORMA ANDROID

Por

ANDRÉ LUIZ LEHMANN

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

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

Blumenau, 09 de Dezembro de 2010

Dedico este trabalho àqueles que não têm medo dos seus sonhos, e nem têm medo do gigante monstro de biscoito.

AGRADECIMENTOS

Agradeço inicialmente aos meus pais, que a 24 anos me aturam, me ensinam, me educam, me guiam e me incentivam a sempre perseguir os meus sonhos, mesmo que isso me leve para longe deles.

Agradeço também a minha irmã, que além de me aturar por todos os seus 20 anos de vida, me traz alegria e leveza de ser.

Agradeço também aos meus nobres amigos, que sempre estão dispostos a jogar uma boa partida de War, Monopoly ou mesmo um simples Uno, rir de nossas situações cotidianas, discutirmos sobre carreira profissional, carreira acadêmica, política, religião e todos os assuntos que causam intrigas (incluindo sistemas operacionais).

Agradeço a alguns professores que tive anteriormente à graduação, como os professores Márcio e Sheila, as pessoas que me fizeram gostar da matemática por detrás do mundo, professor Sílvio de geografia, que era acima de tudo um pensador, professor Renato, por sua disciplina imposta. Obrigado, mesmo que vocês não mais se lembrem de mim, por suas contribuições ao meu caráter.

Agradeço também aos professores da graduação que na medida em que suas atribuições permitiam, me auxiliavam em meus questionamentos, quase sempre extracurriculares, e me incentivavam na pesquisa.

Pessoas a agradecer eu teria aos montes, pois esse momento só foi possível pela contribuição que muitas pessoas deram à minha vida, mas vou me conter aqui, pois todos estão assíduos pelo texto da monografia, e não nas lembranças de um jovem estudante.

A desobediência é uma virtude necessária à criatividade

Raul Seixas

RESUMO

Este trabalho apresenta o desenvolvimento de uma extensão da ferramenta *SipDroid* (comunicador VoIP para a plataforma Android), que possibilita a criptografia dos dados de voz trafegados durante uma conversação VoIP. Apresenta também o desenvolvimento de uma extensão do protocolo ZRTP que implementa o algoritmo de Diffie-Hellman com curvas elípticas para a troca das chaves criptográficas para a utilização do algoritmo simétrico AES.

Palavras-chave: VoIP. Android. Criptografia. Diffie-Hellman. AES. Curvas elípticas.

ABSTRACT

This paper presents a development of an extension of the tool SipDroid (VoIP communicator for the Android platform) which enables data encryption of voice traffic during a VoIP conversation. Also presents a development of an extension of ZRTP protocol than implements the Diffie-Hellman elliptic curves algorithm to exchange cryptographic keys for use the symmetric algorithm AES.

Key-words: VoIP. Android. Cryptography. Diffie-Hellman. AES. Elliptic curves.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

3G - 3º *Generation*

AES - *Advanced Encryption Standard*

ATM - *Asynchronous Transfer Mode*

DES - *Data Encryption Standard*

EDGE - *Enhanced Data rates for* GSM *Evolution*

EUA - Estados Unidos da América

GF - *Galois Field*

GSM - *Global System for Mobile communication*

HTTP - *HyperText Transfer Protocol*

IBM - *International Business Machines*

IETF - *Internet Engineering Task Force*

IP - *Internet Protocol*

ITU-T - *Internacional Telecommunication Union Telecommunication Standardization*

GNU - GNU *is Not Unix*

LGPL - *Lesser General Public License*

MDCP - *Media Device Control Protocol*

MGCP - *Media Gateway Control Protocol*

NIST - *National Institute of Stanrdars and Technology*

NSA - *National Security Agency*

PRNG - *Pseudo-Random Number Generator*

QoS - Qualidade de Serviço

RFC - *Request For Comments*

RNG - *Random Number Generator*

RSA - Ron Rivest, Asi Shamir e Len Adleman

RTCP - RTP *Control Protocol*

RTP - *Real time Transfer Protocol*

SAS - *Short Authentication String*

S-Box - *Substitution Box*

SDK - *Software Development Kit*

SGCP - *Simple Gateway Control Protocol*

SIP - *Session Initiation Protocol*

SMTP - *Simple Mail Transfer Protocol*

TCP - *Transmission Control Protocol*

TGW - *Trunking Gateway*

UC – *Use Case*

UDP - *User Datagram Protocol*

VoIP - *Voice over* IP

ZRTP4J - ZRTP *for Java*

SUMÁRIO

1 INTRODUÇÃO

Se comunicar e estar comunicável é imprescindível nos dias atuais, onde a cada dia mais decisões são tomadas em cada vez menos tempo por mais pessoas. A invenção da comunicação móvel revolucionou a sociedade. Ela permitiu que em qualquer lugar uma pessoa se tornasse acessível e pudesse decidir mais rápido. Esta nova "era do celular", que iniciou na década de 80, baseou-se em protocolos de comunicação proprietários de empresas de telefonia (MALUCHE, 2008, p. 16). Neste mesmo período, a Internet crescia e evoluía como forma de comunicação.

Estas duas formas de comunicação vieram a convergir no final do século XX, quando os dispositivos móveis puderam acessar a Internet e oferecer à população alguns de seus serviços. A disponibilização da Internet via dispositivos móveis ainda não tinha grande apelo consumista, tanto pela Internet não ser tão onipresente quanto nos dias atuais, quanto pela própria tecnologia de comunicação utilizada nos dispositivos. As tecnologias de comunicação não possuíam uma grande largura de banda, por motivos tecnológicos e mercadológicos, mas com o advento da Internet e a geração de conteúdo para a mesma, todo serviço que oferecesse conexão a Internet precisava adequar-se a esta nova realidade (MALUCHE, 2008, p. 79). Assim, no início do século XXI foi lançada a 3ª geração de dispositivos móveis (3G), que oferecia largura de banda adequada a utilização de serviços na Internet (MALUCHE, 2008, p. 66).

Com o advento da Internet nos dispositivos móveis, algumas tecnologias existentes até então somente na Internet, puderam nascer e proliferar nos dispositivos móveis. Uma destas tecnologias foi o tráfego de voz sobre *Internet Protocol* (IP), denominado VoIP (*Voice over IP*).

A tecnologia VoIP permite que dois ou mais terminais comuniquem-se por voz de forma muito mais barata e de maior qualidade que a telefonia convencional (ACMA, 2009). Isso acontece porque ele utiliza toda a infraestrutura da Internet para o tráfego das informações e o custo de tráfego na Internet é bem menor que o custo para trafegar dados nas antigas redes de telecomunicação. Mas uma característica muito difundida nos protocolos das redes de celular e não disponível em todas as implementações de VoIP é o sigilo dos dados trafegados, como em algumas versões da tecnologia *Global System for Mobile communication* (GSM), onde os dados são trafegados de forma sigilosa através do algoritmo A5 (BIRYUKOV et al., 2000). Este sigilo tem como objetivo dificultar que um agente externo à

comunicação possa ler e entender os dados ali trafegados, permitindo, desta maneira, certo grau de privacidade aos interlocutores.

Visto o acima, o presente trabalho propõe-se a desenvolver uma comunicação através da tecnologia VoIP, de forma segura, para a plataforma móvel Android. A plataforma Android inclui: sistema operacional (baseado em Linux), aplicações de *middleware* e um conjunto de aplicações de uso geral.

Esta plataforma disponibiliza uma máquina virtual para a execução das suas aplicações, a Dalvik Virtual Machine, especialmente projetada para a execução em dispositivos móveis. A plataforma oferece toda uma gama de aplicações de *middleware* para as principais funcionalidades de um dispositivo móvel, tal como conexão *WiFi*, *Bluetooth*, 3G, navegador integrado, suporte aos principais formatos de mídia e suporte a gráficos de alto desempenho.

A segurança dos dados trafegados será oferecida por uma implementação de criptografia, que será realizada através do protocolo *Real-time Transport Protocol* (RTP). O protocolo RTP tem como algumas de suas características: detecção de perda, temporização, identificação de conteúdo e independência da camada de transporte, podendo ser *User Datagram Protocol* (UDP) ou *Transmission Control Protocol* (TCP).

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho foi o desenvolvimento de uma aplicação para a plataforma Android que seja capaz de gerar um canal seguro de comunicação VoIP entre os terminais.

Os objetivos específicos deste trabalho podem ser sumarizados da seguinte maneira:

a) realizar comunicação criptografada, através de RTP;
b) disponibilizar um aplicativo *Softphone* na plataforma Android utilizando RTP criptografado.

1.2 ESTRUTURA DO TRABALHO

O capítulo 2 apresenta os assuntos relacionados ao trabalho, tais como: a tecnologia

VoIP, o protocolo *Session Initiation Protocol* (SIP), a plataforma Android, o protocolo RTP, manutenção de sigilo em comunicação e trabalhos correlatos. No capítulo 3 é descrito o desenvolvimento da nova versão da ferramenta. Por fim, o capítulo 4 traz as conclusões do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

A seguir são apresentados os assuntos: a tecnologia VoIP, o protocolo SIP, a plataforma Android, manutenção de sigilo em comunicação e trabalhos correlatos.

2.1 VOIP

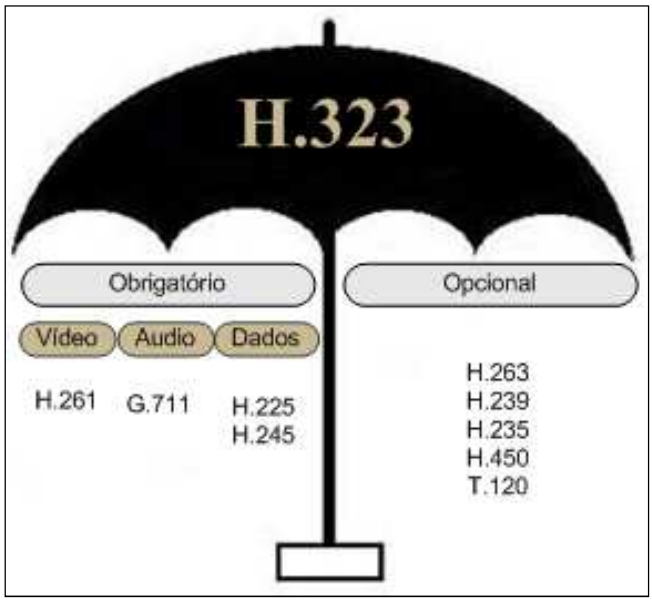
"Voz sobre IP (VoIP) é um conjunto de tecnologias que usam a internet ou redes IP privadas para a comunicação de voz, substituindo ou complementando os sistemas de telefonia convencionais." (AGÊNCIA NACIONAL DE TELECOMUNICAÇÕES, 2010).

Sheppard (2007, p. 25) diz que os benefícios da comunicação VoIP superam, e muito, os seus custos. Além da economia nos custos das ligações internacionais e interurbanos, Sheppard (2007, p. 29) diz que a tecnologia VoIP proporciona economia no *hardware* de telefonia convencional, ganhos de produtividade, garante uma maior proximidade com o cliente da empresa que opta pelo VoIP, através de serviço customizados como vídeo-conferência, chamada a três, além de ganhos na convergência de aplicativos que rodam em cima do VoIP.

A tecnologia VoIP consiste, basicamente, na troca de pacotes de mídia entre dois ou mais dispositivos. Esta troca de pacotes é comumente realizada pelo protocolo *Real-time Transport Protocol* (RTP). Mas, antes da troca efetiva de pacotes de mídia é necessário que todos os dispositivos de uma chamada VoIP estejam em sincronia quanto a forma de transmissão, sinalização da transmissão e várias outras características de uma comunicação multimídia. Esta sinalização se dá através de protocolos de sinalização. Dentre estes protocolos, destacam-se os protocolos H.323, SIP, *Media Gateway Control Protocol* (MGCP) e Megaco/H.248.

O protocolo H.323 faz parte da família de recomendações H.32x da *International Telecommunication Union Telecommunication Standardization sector*, que pertence a família H da ITU-T, e que trata de "Sistemas Audiovisuais e Multimídia" (Leopoldino e Medeiros, 2001). A recomendação H.323 visa especificar sistemas de comunicação multimídia em redes baseadas em pacotes e que não provêem uma Qualidade de Serviço (QoS) garantida. A recomendação H.323 é completamente independente da tecnologia de rede adotada, sendo

capaz de operar tanto em *Ethernet, Fast Ethernet, Token Ring* ou qualquer outra tecnologia na cada de enlace. A recomendação H.323 é comumente relacionada a um guarda-chuva de especificações, pela sua natureza abrangente e modular.



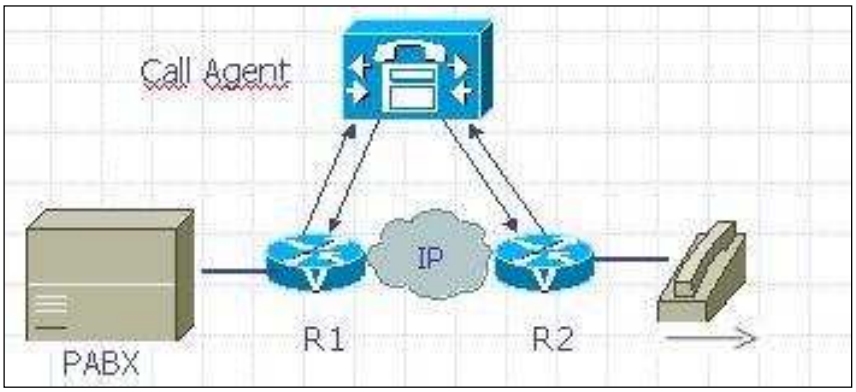
Fonte: Fundação para a Computação Científica Nacional (2004).
Figura 1 – Guarda-chuva H.323

A Figura 1 mostra alguns dos padrões que são abrangidos pela recomendação H.323. Para um *terminal* ser aderente a recomendação H.323, somente é necessário que ele implemente o padrão de áudio G.711 (Fundação para a Computação Científica Nacional, 2004), sendo qualquer outro padrão de áudio opcional. Caso o *terminal* também implementa a transmissão de vídeo ou dados, ele deve obrigatoriamente implementar os padrões H.261 (para áudio) e H.225 e H.245 (para dados). Por suas características modulares, a recomendação H.323 é muito flexível quanto aos participantes de uma sessão VOIP. Mesmo participantes que não possuam canais de vídeo e dados podem participar de uma conversão VOIP, pois eles compartilharão o canal de áudio de forma transparente a todos os participantes. A recomendação H.323 também prevê interoperabilidade de redes, onde os participantes não necessitam estar numa mesma tecnologia de rede, podendo um estar em uma *Local Area Network* (LAN) enquanto os demais participantes podem estar numa rede telefônica pública.

O protoclo MGCP foi proposto em outubro de 1999 através da RFC 2705 (Internet Engineering Task Force, 1999), e a sua arquitetura foi definida na RFC 2805, no ano 2000 (Internet Engineering Task Force, 2000). Ele é o sucessor do *Simple Gateway Control Protocol* (SGCP), proposto inicialmente para interoperar com o protocolo SIP.

O MGCP é um protocolo de sinalização VOIP, assim como o SIP. Possui um conjunto

básico de instruções de comunicação, com comandos para criar, modificar e destruir conexões e requisitar e informar notificações. Todas essas instruções são gerenciadas pelo *Call Agent*, como ilustra a Figura 2 abaixo:

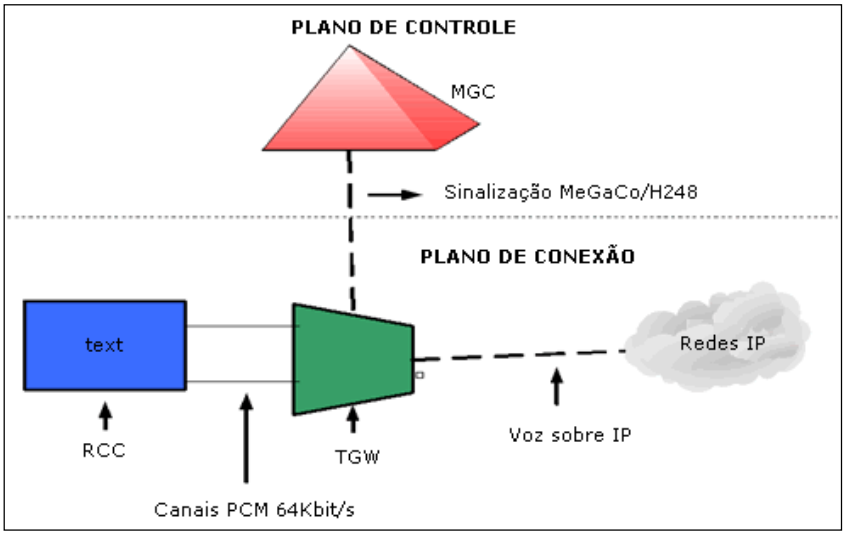


Fonte: Rezende (2004).

Figura 2 – Comunicação MGCP

O protocolo Megaco/H.248 foi desenvolvido em conjunto entre o *Internet Engineering Task Force* (IETF) e o ITU-T. Ele é uma evolução do MGCP do IETF, mantendo algumas características do mesmo, mas adicionando novas habilidades advindas do *Media Device Control Protocol* (MDCP), do ITU-T. Este protocolo é descrito tanto no RFC 3523 do IETF quanto na recomendação H.248 do ITU-T.

O Megaco/H.248 proporciona a intercomunicação entre redes IP e redes telefônicas convencionais. Isso é feito através do *Trunking Gateway* (TGW), que faz a ligação entre as redes públicas de telefonia e a rede IP, *Asynchronous Transfer Mode* (ATM) ou Frame Realy.



Fonte: Oliveira (2006).

Figura 3 – Estrutura simplificada do Megaco/H.248

A estrutura de uma comunicação Megaco/H.248 (Figura 3) é composta de um Plano de Controle (constituído de um *Media Gateway Controller*, que é o responsável pela sinalização

da comunicação) e um Plano de Conexão (constituído do canal de tráfego de dados da comunicação). O Plano de Controle é responsável pela troca de sinalizações e mensagens com outras redes e protocolos, convertendo as mensagens para comandos Megaco/H.248. O Plano de Conexão recebe os comandos Megaco/H.248 para criar e destruir as entidades do protocolo. Não é necessário que o Plano de Conexão esteja fisicamente próximo ao Plano de Controle, visto que ele só necessita dos comandos Megaco/H.248. Outra atribuição do Plano de Conexão é a conversão da mídia de diferentes tipos de rede para a rede IP.

2.2 SIP

De acordo com Internet Engineering Task Force (2010), SIP é um protocolo baseado em texto, similar ao *HyperText Transfer Protocol* (HTTP) e o *Simple Mail Transfer Protocol* (SMTP), para a iniciação de sessões de comunicação interativas entre usuários.

Esse é atualmente o protocolo de sessão mais utilizado dentro da tecnologia VoIP. Ele é o responsável por estabelecer, modificar e terminar uma chamada VoIP entre dois usuários.

Sua arquitetura é baseada no modelo de cliente-servidor onde os clientes iniciam uma chamada e o servidor responde às chamadas. É definido na *Request For Comments* (RFC) 3261 do IETF (2010).

Parafraseando Colcher et al. (2005, p. 189), SIP é um elemento que pode ser usado em conjunto com outros protocolos e componentes na construção de uma arquitetura multimídia completa, sendo a mesma simples, extensível, utilizando protocolos já existentes na Internet e dando preferência a serviços providos de forma fim-a-fim, poupando recursos onde possível.

Similar ao HTTP, no SIP existe um conjunto limitado de métodos para realizar todas as suas operações: são 6 métodos principais, apresentados no Quadro 2.

Comando	Função
INVITE	Iniciar uma chamada
ACK	Confirmação de uma operação
BYE	Término e transferência de uma chamada
CANCEL	Cancela pesquisa e sinal de toque
OPTIONS	Requisição das características suportadas por outro participante
REGISTER	Registro de um cliente no servidor *Registrar*

Fonte: adaptado de Hommerding e Mansur (2006, p. 14).

Quadro 1 – Comandos SIP

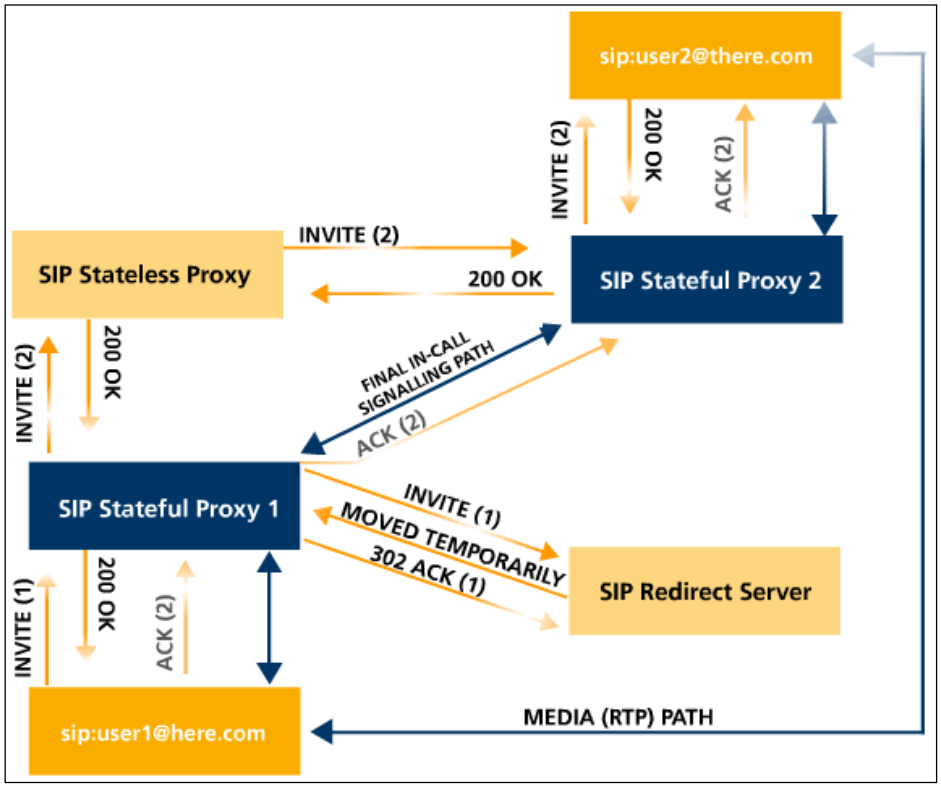
Existem também 6 categorias de respostas possíveis para cada troca de mensagem entre um cliente e servidor SIP, apresentadas no Quadro 3.

Categoria (prefixo do código)	Função
Provisório (1xx)	Respostas de informações
Sucesso (2xx)	Mensagem recebida, entendida e aceita
Redirecionamento (3xx)	É necessária alguma decisão pelo cliente SIP
Erro no cliente (4xx)	Resposta de requisições falhas
Erro no servidor (5xx)	Mensagem de falha no servidor
Falha global (6xx)	Mensagem de falha geral do sistema

Fonte: adaptado de Hommerding e Mansur (2006, p. 14).

Quadro 2 – Respostas SIP

Na Figura 4 tem-se uma representação gráfica exemplificando uma chamada VoIP através do protocolo SIP.



Fonte: Session initiation protocol (2010).

Figura 4 – Comunicação SIP

Na Figura 4 é mostrado todo o processo de iniciação de sessão entre todos os envolvidos numa chamada VoIP através do SIP. O processo inicia-se com o comando `INVITE` partindo de `sip:user1@here.com` para `sip:user2@here.com`, passando pelos servidores que fazem o papel de *proxy*, sendo a chamada redirecionada através da resposta `302 moved temporarily` do servidor SIP *Redirect*, e alcançando o `sip:user2@here.com`, o qual aceita a chamada e responde com um comando de *ACKnowledge* (ACK). Assim que `sip:user1@here.com` recebe a confirmação, ele pode iniciar uma comunicação através do `Media (RTP) path`, e desta forma, realizar a chamada com `sip:user2@here.com` através do protocolo RTP.

2.3 RTP

De acordo com Almeida (2003), RTP é um protocolo utilizado para o transporte de mídias contínuas de tempo real, tanto em conexões ponto-a-ponto quanto em conexões *multicasting*. O protocolo RTP não fornece garantia quanto à reserva de recurso e nem quanto à qualidade de serviço (QoS). Para amenizar os problemas por ventura gerados por essas características, é utilizado o *RTP Control Protocol* (RTCP), que faz o envio periódico de pacotes de controle para todos os participantes na sessão, podendo coletar informações sobre o estado da conexão de cada participante da sessão.

Como dito por Bruno, Duarte e Roth (2006) o protocolo RTP tem capacidade para trabalhar em redes com terminais que possuem diferentes larguras de banda de acesso. Isso é feito através dos Misturadores (*Mixers*). Os *mixers* ficam localizados próximos aos pontos de menor largura de banda. Os *mixers* recebem e redistribuem novos pacotes de transmissão, podendo com isso sincronizar e misturar as diversas fontes de *stream* RTP, desonerando a carga da rede a partir do ponto onde o *mixer* está instalado. O protocolo RTP trabalha também com certas peculiaridades de redes, como a presença de um *firewall* e a mudança esporádica de protocolo (por exemplo, a mudança do protocolo TCP para o protocolo UDP). Para essas peculiaridades, o RTP disponibiliza o mecanismo de Tradutores (*Translators*). Um *translator* é responsável por receber e traduzir as mensagens em pacotes compatíveis com a parte da rede que o sucede. Este mecanismo permite que se tenha controle sobre o tráfego da rede que trafega sobre o protocolo RTP, além de permitir que cada participante da sessão RTP possa estar numa rede com um protocolo de transmissão diferente dos demais, permitindo certo grau de independência.

Outra característica do protocolo RTP é que cada mídia diferente exige uma sessão RTP diferente. Então, numa transmissão de áudio e vídeo, haverá no mínimo duas sessões RTP distintas: uma exclusivamente para a transmissão de áudio e outra para a transmissão de vídeo. Isto permite que cada tipo de mídia tenha transmissão com configurações diferentes, para clientes diferentes, proporcionando um QoS para cada tipo de mídia.

2.4 ANDROID

Android é uma pilha de software para dispositivos móveis que inclui um sistema operacional, *middleware* e aplicações básicas para a operacionalidade do sistema (ANDROID DEVELOPERS, 2010).

O ambiente Android foi inicialmente desenvolvimento pela Android Inc., sendo esta adquirida pela Google Inc. no ano de 2005 (ELGIN, 2005). No ano de 2007 o até então projeto Android passou às mãos do *Open Handset Alliance*, grupo formado por grandes empresas do setor móvel com o objetivo de desenvolver padrões abertos para dispositivos móveis (OPEN HANDSET ALLIANCE, 2007).

De acordo com Google (2008), o primeiro celular usando o Android como sistema operacional foi o T-Mobile G1, no ano de 2008. Desde lá, mais de 115 aparelhos telefônicos, e mais de 50 dispositivos móveis não-telefônicos também passaram a utilizá-lo (GOOGLE AND BLOG, 2009).

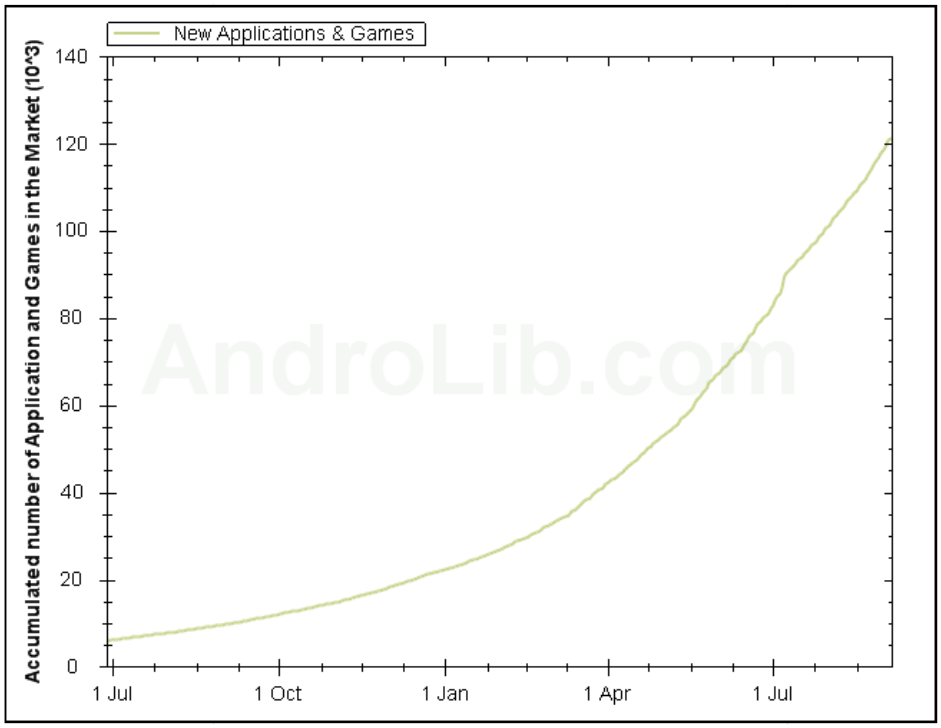
Segundo Hamblen (2009), a plataforma Android deve ser, no ano de 2012, a plataforma de *smartphones* com o maior mercado nos Estados Unidos, com um percentual de 14% do mercado. Isso a coloca a frente da plataforma *iPhone*, *Windows Mobile* e *Black Berry* na preferência dos usuários norte-americanos.

A principal linguagem de programação para a plataforma Android é a linguagem Java, mas por ser compilado e executar na máquina virtual Dalvik, o ambiente Android possui certas particularidades. O *Software Development Kit* (SDK) da plataforma Android é diferente do ambiente de programação Java usualmente encontrado no ambiente *desktop*. O SDK da plataforma Android teve todas as suas classes e componentes criados especificamente para a plataforma móvel, sem reaproveitamento de código da plataforma Java para *desktop*.

Somente o código-fonte de ambas as plataformas são idênticos, pois o código intermediário da plataforma Android difere da plataforma Java convencional em função da arquitetura de execução do código. Na plataforma Android é executado numa máquina de registradores, a máquina virtual Dalvik, enquanto a plataforma Java é executada numa máquina de pilha, a máquina virtual Java convencional. Esta decisão arquitetural foi tomada em função do *hardware* para o qual a máquina virtual Dalvik é projetada, os *smartphones*.

A disponibilidade de aplicativos na plataforma Android cresceu de forma sustentada e vigorosa nos últimos anos. De acordo com AndroidLib (2010), no mês de agosto de 2010 a principal fonte de aplicativos, a Android Market, possuía mais de 100 mil aplicativos

disponibilizados para *download*, tanto pagos quanto gratuitos (Figura 4).



Fonte: AndroidLib (2010).

Figura 5 – Estatística Android Market

No meio desse universo de *softwares* alguns ganham destaque, como o Sipdroid, um comunicador VOIP baseado no protocolo SIP para a plataforma Android. O Sipdroid é um projeto de *software* livre, licenciado de acordo com termos da licença GNU *General Public License*, versão 3. Projetado para funcionar primariamente com o servidor pbxes.org (Sipdroid, 2009), hoje em dia já possui suporte para vários servidores, como o sip2sip.info e o 3CXPhone, da empresa 3CX Inc. A Figura 6 apresenta a interface gráfica do SipDroid.

Fonte: Sipdroid (2010).

Figura 6 – Interface gráfica do Sipdroid e Interface de configuração

O Sipdroid possui suporte tanto a áudio quanto a vídeo, sendo o suporte a vídeo bem restrito, permitindo somente o envio de vídeo através de *streaming* (Sipdroid, 2009). Atualmente, o Sipdroid possui suporte a 10 diferentes formatos de áudio e a 1 formato de vídeo. Possui suporte a transmissão tanto de pacotes *Transmission Control Protocol* (TCP) quanto *User Datagram Protocol* (UDP), tanto em redes *Ethernet* quanto redes 3G e *Enhanced Data rates for GSM Evolution* (EDGE), conforme mostra a Figura 6.

Uma característica faltante ao Sipdroid, e que é requisitado por seus usuários, é a capacidade de criptografia do canal de comunicação, conforme requisição em Sipdroid (2010). Este já é um pedido antigo, datando de 2009, mas não prontamente implementado pelo projeto Sipdroid, justamente devido a complexidade que envolve o assunto, considerando que no atual estágio, o Sipdroid já é compatível com todo e qualquer comunicador SIP que implemente o protocolo SIP mesmo que de forma somente satisfatória, tendo assim que se preocupar com compatibilidade entre comunicadores.

2.5 SIGILO

Quando é necessário manter sigilo sobre as informações, a maneira mais comum é simplesmente escondê-las de outras pessoas. Mas, dependendo do ambiente é necessário passar a informação à outra pessoa. Quando o meio pelo qual a informação passa é público ou é de terceiros (como exemplo, cita-se uma sala lotada de pessoas, o ambiente de trabalho, uma empresa de entregas) é necessário que a mensagem seja cifrada numa forma ilegível a outros, e que somente o verdadeiro destinatário possa compreendê-la.

Como define Moreira (2002), encriptação consiste na aplicação de um algoritmo aos dados de forma que eles tornem-se ilegíveis, e para a recuperação dos dados é necessário ao destinatário o conhecimento prévio do algoritmo de decriptação.

Não existe encriptação 100% eficaz, visto que sendo do conhecimento de um eventual atacante alguma mensagem exemplo, sua contraparte cifrada e conhecendo o algoritmo, é possível que qualquer outra mensagem possa ser quebrada, pela utilização da técnica da força bruta. Para a correta escolha do algoritmo mais indicado para determinada situação, devem ser levados algumas características do algoritmo: força, tempo para encriptação, tempo para decriptação, capacidade computacional disponível para encriptar/decriptar dados, como

demonstram Olson e Yu (2000).

2.5.1 Criptografia de chave simétrica

Conforme Burnett e Paine (2002, p. 12 a 18), além do conhecimento do algoritmo utilizado, para uma segurança adicional é necessário um número secreto, denominado chave, que serve de maneira análoga a chave de uma fechadura real. O conteúdo encriptado somente é legível com a utilização do algoritmo de decriptação correto e a chave geradora. Esta forma de criptografia é denominada de criptografia de chave simétrica, justamente pelo simetrismo exigido entre as chaves nos processos de encriptação e decriptação dos dados. Dessa maneira, o segredo está contido na chave utilizada para encriptar/decriptar os dados, e a chave é simplesmente um número.

Este número deve ser um número aleatório e suficientemente grande para dificultar qualquer forma de ataque à criptografia. Para averiguar a aleatoriedade da chave escolhida, são utilizadas várias técnica, entre as quais se destaca os testes Kolmogorov-Smirnov, Chi-Quadrado, teste de Auto-Correlação, *Gap Test* e *Poker Test*. Estes teste procuram definir se uma determinada sequência de bits está distribuída de forma uniforme, sem repetições e nem correlações entre as partes da sequência. Para a obtenção do número aleatório utilizado como chave existem duas maneiras: através de um Gerador de Número Aleatório (*Random Number Generator* - RNG), um dispositivo físico capaz de gerar números verdadeiramente aleatórios, ou através de um Gerador de Número Pseudo-Aleatório (*Pseudo-Random Number Generator* - PRNG), um algoritmo capaz de gerar números que passem em testes de aleatoriedade, mas por serem repetíveis (através de uma mesma entrada de dados produzem o mesmo número) são denominados pseudo-aletórios (BURNETT e PAINE, 2002).

Mas de nada basta uma chave segura e forte, se for usado um algoritmo fraco e não-confiável. Como argumentado por Burnett e Paine (2002, p. 18 a 21), um algoritmo de criptografia público, que foca a força da chave utilizada para criptografar, e não numa implementação específica do mesmo, é mais forte que um algoritmo fechado, onde a força do mesmo está focada em pequenos (e nem tão pequenos) detalhes da implementação do mesmo, um algoritmo que tenta ser forte através da obscuridade de sua implementação. E foi na década de 70 que a *International Business Machines* (IBM) juntamente ao *National Security Agency* (NSA) desenvolveram o algoritmo *Data Encryption Standard* (DES), que se tornou livremente disponível a partir da sua publicação. A partir da sua publicação, o DES foi

amplamente estudado, e era consenso entre especialistas que o mesmo não havia falhas e assim ele se tornou algoritmo padrão de criptografia por vários anos.

Em função da evolução do poder computacional, que estava seguindo a Lei de Moore, com o número de transistores duplicando a cada 18 meses, o DES e a sua chave de 56 bits estavam perigosamente próximos de serem quebrados num tempo hábil. Isso aconteceu durante a década de 1990, com a prova cabal sendo a quebra de uma chave DES em 1999 num período de 24 horas (Burnett e Paine, 2002, p. 40). Como substituto dos DES, foi criado o *Triple* DES, que é a execução seqüencial do algoritmo DES 3 vezes nos dados. Isto deu uma sobrevida ao DES, mas o algoritmo resultante ainda possuía fraquezas, além de levar 3 vezes mais tempo que o antigo DES, que já era um algoritmo lento de criptografia (Burnett e Paine, 2002, p. 40 e 41). Foi em 1997 que a *National Institute of Standars and Technology* (NIST) iniciou um projeto para a concepção de um novo padrão de criptografia. O projeto era aberto ao público em geral, e promovia que fosse enviado ao instituto qualquer algoritmo criptográfico, com a condição que o autor abrisse mão de qualquer direito de propriedade intelectual para o mesmo. Deste concurso foi escolhido o algoritmo Rijndael, criado por Vincent Rijmen e Joan Daemen. Este algoritmo ficou conhecido como *Advanced Encryption Standard* (AES).

De acordo com *National Institute of Standards and Technology*, o AES é um cifrador de bloco simétrico, capaz de utilizar chaves criptográficas de 128, 192 e 256 bits para encriptar e decriptar blocos de 128 bits. Desde 2001, quando o algoritmo Rijndael foi escolhido como vencedor do concurso, o AES é o algoritmo de encriptação padrão do governo dos Estados Unidos da América (EUA). O AES é uma versão modificada do Rijndael original, sendo que esse permitia vários outros tamanhos para a chave criptográfica, assim como outros tamanhos para o bloco de encriptação, mas estes estão fora do escopo do AES.

A força do algoritmo AES está no tamanho da equação que o representa. De acordo com Courtois e Pieprzyk (2002), uma chave criptográfica AES de 128 bits de tamanho, que possui um universo de 2^{128} possibilidades de chaves distintas, pode ser reduzida a uma equação de 2^{50} termos, o que lhe garante um alto nível de segurança. A chave de 256 bits possui uma equação reduzida de 2^{70} termos (Ferguson, Schroeppel e Whiting), o que dificulta ainda mais qualquer forma de ataque de força-bruta.

As operações do AES são feitas em elementos de campo finito que possuem as operações de soma e multiplicação. Estas operações se comportam de maneira diferente dos seus pares em números convencionais. Computacionalmente, a operação de adição em

campos finitos é realizada através de uma operação XOR. Já a operação de multiplicação ocorre sobre um Corpo de Galois GF(2^8), definido conforme a figura 8:

$$m(x) = x^8 + x^4 + x^3 + x + 1.$$

Fonte: *NIST* (2001).
Quadro 3 – Corpo de Galois GF(2^8)

A operação de multiplicação ocorre como uma multiplicação de matrizes normal, efetuando o módulo de GF(2^8), definido no Quadro 1. Com a operação de módulo em GF(2^8) é garantido que o resultado da operação terá grau menor que 8, e dessa forma poderá ser representado por um byte (NIST, 2001).

O algoritmo AES é um algoritmo iterativo, composto de 4 operações básicas:

- AddRoundKey;
- SubBytes;
- ShiftRows;
- MixColumns.

A operação de `AddRoundKey` faz uma adição de campo finito entre a chave da rodada e o bloco a ser criptografado, comumente denomidado *state*. Para a obtenção da chave da rodada, antes mesmo da primeira execução do `AddRoundKey`, é feita uma expansão da chave criptográfica passada ao algoritmo, para que de acordo com o tamanho da chave criptográfica possa ser gerada uma chave expandida que comporte todas as rodadas necessárias para a execução do AES. Após a expansão da chave, para cada rodada do AES é escolhida uma chave da rodada, a qual será efetuada a operação de adição com o *state*, gerando um novo *state* intermediário.

A operação de `SubBytes` realiza uma permutação entre o *state* e uma tabela de substituição, conhecida como S-Box. A S-Box possui todas as possibilidades para uma palavra na representação hexadecimal. Desta forma, todo e qualquer byte de *state* pode ser transformado através desta tabela. A S-Box é formada pela multiplicativa inversa do campo finito GF(2^8), com o elemento `00` sendo mapeado para ele mesmo, e aplicando a transformação afim mostrada no Quadro 4:

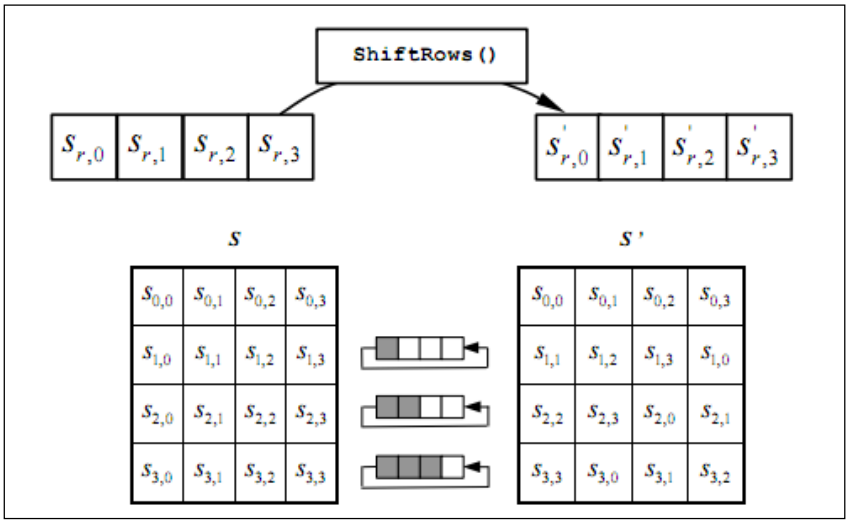
$$b_i' = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i$$

Fonte: NIST (2001).
Quadro 4– Transformação afim

A S-Box é uma tabela fixa e invariável, e de acordo com Fergunson, Schroeppel e Whiting, é está natureza imutável que faz da S-Box um ponto inicial para um ataque contra o

AES. A equação para descrever o comportamento da S-Box com chaves criptográficas de vários tamanhos pode ser simplificada, e isto reduz consideravelmente o tempo para um ataque.

A operação `ShiftRows` faz uma operação de *shift* em cada linha de *state*. A operação de *shift* faz o deslocamento cíclico dos bytes dentro de um vetor. O efeito desta operação é mover os bytes para posições "menores" no vetor, enquanto que os bytes que já se encontram nas "menores" posições são ciclicamente movidos para as posições maiores. A Figura 7 ilustra uma execução da operação `ShiftRows`:



Fonte: *NIST* (2001).

Figura 7 – Operação `ShiftRows`

A operação `MixColumns` utiliza cada vetor de *state* como um elemento de uma operação de multiplicação em corpos finitos para um polinômio *a(x)* definido no Quadro 5, e após a operação de multiplicação é realizada a operação de módulo sobre o polinômio definido por $x^4 + 1$. O resultado dessa operação é atribuído ao novo estado do vetor em *state.*

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

Fonte: NIST (2001).

Quadro 5 – Polinômio *a(x)*

O número de iterações do algoritmo AES depende unicamente do tamanho da chave criptográfica escolhida, sendo os únicos tamanhos permitidos pela definição do algoritmo, 128, 192 e 256 bits, com o número de iterações de 10, 12 e 14, respectivamente. No Quadro 6 é possível ver o pseudo-código do cifrador AES.

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])

   for round = 1 step 1 to Nr-1
      SubBytes(state)
      ShiftRows(state)
      MixColumns(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```

Fonte: NIST (2001).

Quadro 6 – Pseudo-código do cifrador AES

O decifrador AES consiste também de 4 operações básicas, sendo as mesmas denominadas como inversas das operações básicas do cifrador, exceto a operação de `AddRoundKey`, que se mantém igual entre o cifrador e o decifrador. O Quadro 7 mostra o pseudo-código do decifrador, utilizando as operações inversas mais a operação `AddRoundKey`.

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   for round = Nr-1 step -1 downto 1
      InvShiftRows(state)
      InvSubBytes(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
      InvMixColumns(state)
   end for

   InvShiftRows(state)
   InvSubBytes(state)
   AddRoundKey(state, w[0, Nb-1])

   out = state
end
```

Fonte: NIST (2001).

Quadro 7 – Pseudo-código do decifrador AES

A operação de `InvShiftRows` possui o mesmo comportamento de `ShiftRows`, mas inverte o sentido da rotação, com os bytes "maiores" sendo deslocados para as posições "menores". Já a operação de `InvSubBytes` possui o mesmo mecanismo de `SubBytes`, mas utiliza outra *S-Box*, que é definida pelo mesmo corpo finito de `SubBytes`, GF(2^8), mas aplicando a transformação afim inversa da apresentada no Quadro 2. A operação de `InvMixColumns` efetua a multiplicação em corpos finitos de cada vetor de *state* pelo

polinômio $a^{-1}(x)$ definido no Quadro 8, efetuando em seguida a operação módulo por $x^4 + 1$.

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}.$$

Fonte: NIST (2001).

Quadro 8 – Polinômio $a^{-1}(x)$

2.5.2 Criptografia de chave assimétrica

A força de uma criptografia de chave simétrica está relacionada intimamente com a força da sua chave criptográfica, mas existe um ponto fraco nesta técnica de criptografia: os dois lados da comunicação devem conhecer a mesma chave criptográfica. Isso exige que haja a troca destas chaves, fornecendo assim mais uma grande brecha para o vazamento de informações. Pois, não há como garantir uma troca de chaves seguras, sendo o meio de troca não-seguro, como uma rede de computadores. Mesmo numa comunicação já criptografada, existe a necessidade da troca das chaves para esta primeira criptografia, tornando assim o problema recursivo, pois para tornar o ambiente seguro, utilizo o mesmo mecanismo para a troca da chave criptográfica. Uma boa solução para este problema está na utilização de criptografia de chave assimétrica.

Criptografia de chave assimétrica consiste de um algoritmo de criptografia que utiliza duas chaves distintas: uma para encriptar os dados e outra para decriptar (Burnett e Paine, 2002, p. 74). A criptografia de chave assimétrica também é conhecida como criptografia de chave pública-privada, pois com a existência destas duas chaves, somente é necessário o segredo de uma delas, que é denominada chave privada. A sua contra-parte é chamada de chave-pública pois ela pode ser conhecimento público, sem qualquer ameaça a segurança do conjunto de chaves. Não existe uma ordem explícita sobre qual chave deve ser utilizada para encriptar e qual deve ser utilizada para decriptar. Isso acontece pois as chaves são interligadas matematicamente, através de alguns problemas matemáticos chamados de *funções de via única* (Burnett e Paine, 2002, p. 80), que são funções matemáticas não-verdadeiramente de vias únicas, mas sim que possuem uma porta de interrupção, que é a chave privada.

Os principais problemas matemáticos utilizados para a criptografia de chave pública são: fatoração, logaritmo discreto e curvas elípticas (Burnett e Paine, 2002, p. 83). Cada um desses possui ao menos um conhecido algoritmo de chave pública-privada. O algoritmo RSA, desenvolvido em conjunto por Ron Rivest, Adi Shamir e Len Adleman, utiliza a fatoração de grandes números para possibilitar a criptografia de chave assimétrica. O algoritmo de Diffie-

Helmann, desenvolvido na década de 70 por Whitfield Diffie e Martin Hellman, utiliza logaritmo discreto para a geração das chaves assimétricas. Existe uma variação de Diffie-Hellman que utiliza curvas elípticas para a geração das chaves pública-privada.

Cada chave do algoritmo RSA se compõe de dois elementos-chave: o módulo e o expoente. Para a chave-pública, o valor do módulo é definido pela multiplicação de dois valores primos, usualmente definidos por *p* e *q*, enquanto que o valor do expoente segue a fórmula definida no Quadro 9. Já para a chave-pública, o valor do módulo é o mesmo da chave-privada, mas o valor do expoente é definido pela redução modular da fórmula definida no Quadro 10.

$$MDC(e, (p-1)(q-1)) = 1$$

Fonte: Johnston (2010).
Quadro 9 – Cálculo da chave-pública

$$d = e \pmod{(p-1)(q-1)}$$

Fonte: Johnston (2010).
Quadro 10 – Cálculo da chave-privada

Após a definição das chaves, os valores de *p* e *q* apesar da sua importância extrema, já que são os primos que definem as chaves, não necessitam mais serem armazenados, podendo descartá-los. Para as operação de encriptação e decriptação, cada valor de chave é utilizada na fórmula definida no Quadro 11, com o valor de *k* alterando entre *e* e *d*, dependendo da finalidade, seja ela cifrar ou decifrar o texto-plano, com o valor de *m* sendo o texto de entrada para o algoritmo.

$$c = m^k \pmod{N}$$

Fonte: *Johnston* (2010).
Quadro 11 – Cálculo da cifragem/decifragem

O algoritmo de Diffie-Hellman, usando o problema de logaritmo discreto da matemática, utiliza como fórmula-base a definição em Quadro 12, com *x* sendo o valor da chave e *g* e *n* sendo valores definidos entre as partes participantes da comunicação. Para o algoritmo Diffie-Hellman existem poucas restrições quanto aos números escolhidos:

a) *g* deve ser menor que *n*;
b) *g* deve ser maior que 1.

$$g^x \pmod{n}$$

Fonte: Johnston (2010).
Quadro 12 – Fórmula-base de Diffie-Hellman

Após a definição de um *g* e de um *n* compatíveis com as regras acima expostas, e que seja de conhecimento de todas as partes, cada parte da comunicação deverá definir um valor

para *x*, que será a sua chave-privada e efetuando o cálculo da fórmula-base. De posse do valor obtido, todas as partes trocam os valores, que a partir deste ponto é chamado de chave-pública. Agora, cada participante pode calcular a chave-secreta da comunicação, simplesmente executando a fórmula-base de Diffie-Hellman, mas em vez de usar o *g* escolhido anteriormente, é usado o valor recebido do outro participante da comunicação. A matemática garante que todos os participantes terão a mesma chave-secreta, e com isso poderão se comunicar de forma criptografada, independentemente da quantidade de participantes da comunicação.

Este modelo de troca de chaves possibilita que as chave público-privada sejam etéreas, sem a necessidade de armazenamento e nem o gerenciamento das chaves já utilizadas, além da segurança proporcionada pelo algoritmo, que se baseia num problema tratado pela matemática como de via única, e potencialmente intratável para agentes externos a comunicação.

Outra possibilidade de problema matemático que pode ser utilizado para a criptografia assimétrica é o problema das curvas elípticas. Este problema é uma especialização do problema de logaritmo discreto, e dessa forma, a criptografia de curvas elíptica é baseada no algoritmo de Diffie-Hellman. Para o algoritmo de Diffie-Hellman com curvas elípticas, em vez do problema de logaritmo discreto de Diffie-Hellman formulado no Quadro 12, é utilizada a fórmula de uma curva elíptica qualquer. Genericamente, uma curva elíptica é definida pela fórmula no Quadro 13.

$$y^2 = x^3 + ax + b$$

Fonte: Burnett e Paine (2002).
Quadro 13 – Fórmula de uma curva elíptica

Numa curva verdadeiramente elíptica, dada a soma de um ponto P0 pertencente a curva com outro ponto P1 também pertencente a curva ou mesmo com o próprio ponto P0, o resultado será um outro ponto pertencente a curva elíptica (Burnett e Paine, 2002, p. 96). É esta característica das curvas elípticas que é aproveitada para o algoritmo de Diffie-Hellman com curvas elípticas, já que com a soma de P0 em P0 temos, na realidade, uma multiplicação escalar dos pontos. Dada certa curva elíptica *E*, cada elemento da comunicação define um ponto *P* qualquer na curva e o multiplica por um valor *d*, usualmente um número primo grande, que esteja entre zero e o tamanho do campo finito da curva elíptica. O resultado desta multiplicação escalar é denominado *Q*. A tupla *E*, *P* e *Q* é definida como a chave-pública do participante, enquanto que o escalar *d* é a chave-privada do mesmo.

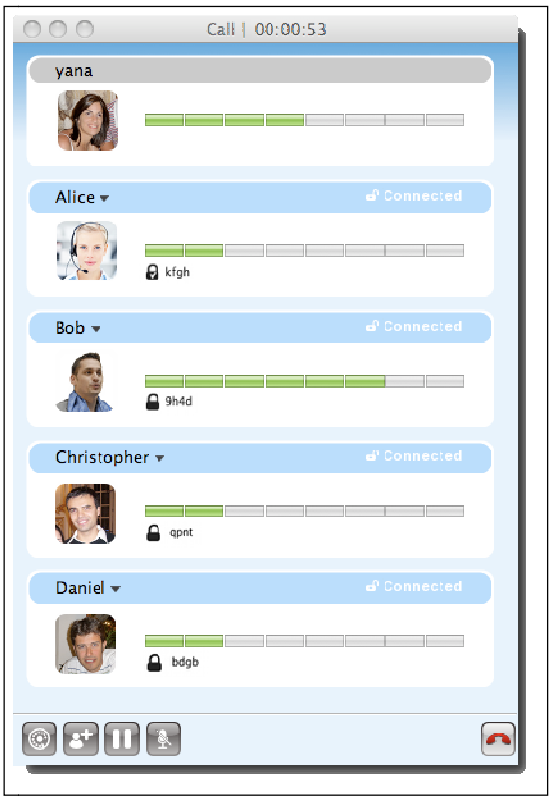
2.6 TRABALHOS CORRELATOS

A seguir são apresentados trabalhos sobre a comunicação dos dados VoIP, as quais são: SIP *Communicator* (SIP-Communicator, 2005) e "VoIP para computação móvel", trabalho de BAZOTTI (2007).

2.6.1 SIP Communicator

Segundo SIP-Communicator.org (2010), o SIP *Communicator* é um *softphone open-source*, distribuído sobre a licença *GNU Lesser General Public License* (LGPL), que permite tanto a transmissão de áudio quanto de vídeo e funcionando também como um trocador de mensagens instantâneas. Ele suporta alguns dos mais populares protocolos de comunicação, como SIP, Jabber, Yahoo! *Messenger*, Bonjour. Possui capacidade de criptografar o canal de transmissão VoIP, através do protocolo ZRTP.

Atualmente, o SIP Communicator é a implementação referência do projeto ZRTP *for* Java (ZRTP4J), que é uma implementação do protocolo ZRTP para a linguagem Java (GNU *Telephony*, 2009). O projeto ZRTP4J proporciona a camada de criptografia para o protocolo SIP, utilizando o algoritmo de Diffie-Helmann para a troca das chaves criptográficas, e usando como algoritmo criptográfico o *Advanced Encryption Standard* (AES) (*Internet Engineering Task Force*, 2010).



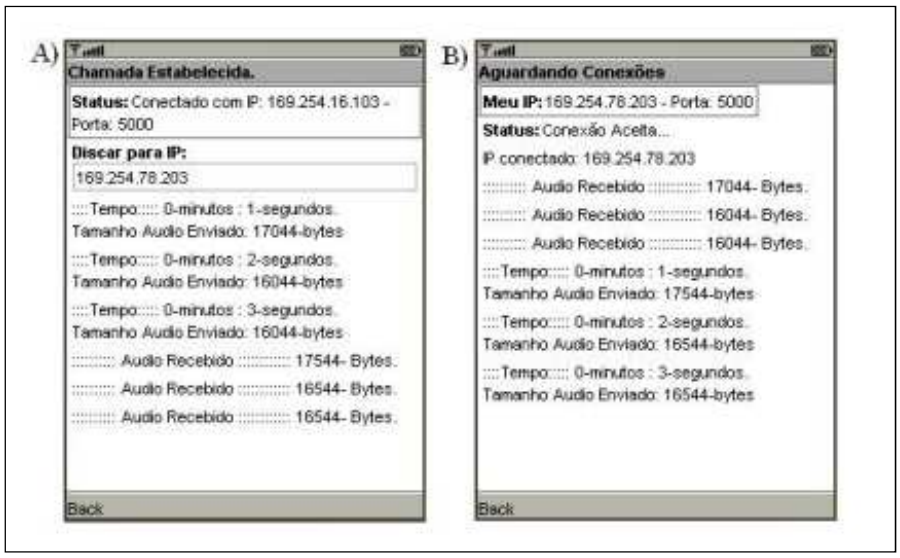
Fonte: SIP-Communicator (2010).

Figura 8 – Comunicação criptografada

Numa comunicação VoIP através do SIP Communicator (Figura 8), na detecção que todas as partes presentes suportam o recurso de criptografia baseado em ZRTP, é criado um SAS para cada parte e todos os participantes da conversação autenticam as demais partes através do reconhecimento da voz.

2.6.2 VoIP para computação móvel

Como descrito em Bazotti (2007), foi desenvolvido uma proposta de comunicação VoIP para dispositivos móveis. Como plataforma de desenvolvimento, foi definido que seria utilizado a versão da plataforma Java para dispositivos móveis, o *Java Mobile Edition* (Java ME). Está plataforma possui uma boa maturidade, com um bom conjunto de bibliotecas para acesso aos serviços oferecidos por uma grande gama de dispositivos, além de proporcionar uma abstração do dispositivo físico, trabalhando somente com objetos abstratos o suficiente para que sejam utilizáveis num grande número de diferentes aparelhos, desde celulares, *smartphones*, *Personal Digital Assistant* (PDA), *tablets* etc.



Fonte: Bazotti (2007).

Figura 9 – Comunicador de Bazotti

O comunicador desenvolvido por Bazotti (Figura 9) foi projetado para funcionar em dois tipos de redes: *wireless* e *Bluetooth*. Estes tipos de rede proporcionam interoperabilidade entre vários tipos de aparelhos distintos, permitindo a comunicação de celulares com *smartphones* ou mesmo com *tablets*. A forma como a comunicação entre as duas partes da comunicação trocavam os pacotes foi através de *sockets* de rede, utilizando como protocolos de transporte tanto TCP quanto UDP.

Para a inicialização de uma comunicação VoIP, Bazotti optou por utilizar um protocolo próprio, com algumas semelhanças ao SIP. Isso proporcionou que fossem criadas instruções mais próximas as características que Bazotti acreditava serem as mais corretas.

Ao final do seu desenvolvimento, Bazotti teve alguns problemas técnicos com a tecnologia Java ME e a integração de uma placa *wireless* com o seu dispositivo *Palm Treo 680*, que era o dispositivo-alvo do seu projeto. Também foram utilizados celulares *Sony Ericsson* para a comunicação *Bluetooth*, e que funcionaram como o esperado. A comunicação entre um computador de mesa com os dispositivos móveis funcionou como o esperado, somente sendo necessários alguns ajustes nas configurações da comunicação.

3 DESENVOLVIMENTO

Neste capítulo são apresentadas as seguintes seções: especificação dos requisitos do software desenvolvido, diagramas de casos de uso, classes, atividades e sequência, implementação do algoritmo de Diffie-Hellman, integração com outros comunicadores que suportam Diffie-Hellman.

Ainda, os resultados obtidos pela comunicação criptografada de dados e as dificuldades encontradas também são relatados.

3.1 REQUISITOS DO SOFTWARE DESENVOLVIDO

Os requisitos funcionais do software são:

a) efetuar a troca de chaves criptográficas através do algoritmo de Diffie-Hellman;
b) gerar as chaves assimétricas usando o problema de logaritmo discreto;
c) realizar a comunicação VoIP através da troca de pacotes RTP criptografados com o algoritmo AES;
d) adicionar ao *Softphone* SIPDroid a capacidade de troca de pacotes criptografados.

Os requisitos não-funcionais do sistema são:

a) utilizar a linguagem *Java*;
b) executar na plataforma *Android*;
c) suportar a biblioteca ZRTP4J versão 1.1;
d) suportar a biblioteca Bouncy *Castle Crypto* versão 1.45.

3.2 ESPECIFICAÇÃO

Para a especificação do software foram utilizados os diagramas da *Unified Model Language* (UML) como caso de uso, diagrama de classe, diagrama de sequência e diagrama de atividade. Para montar a especificação foi utilizada a ferramenta *Enterprise Architect*.

3.2.1 Diagrama de Caso de Uso

A seguir são apresentados os diagramas das três principais fases da comunicação VoIP segura: Construção de chaves, Negociação de chaves e Utilização de chaves, com seus respectivos casos de uso, que representam as funcionalidades da camada criptográfica.

3.2.1.1 Construção de chaves

A Figura 10 apresenta os casos de uso referentes a construção das chaves público-privada, tendo como único ator o próprio *softphone* SipDroid.

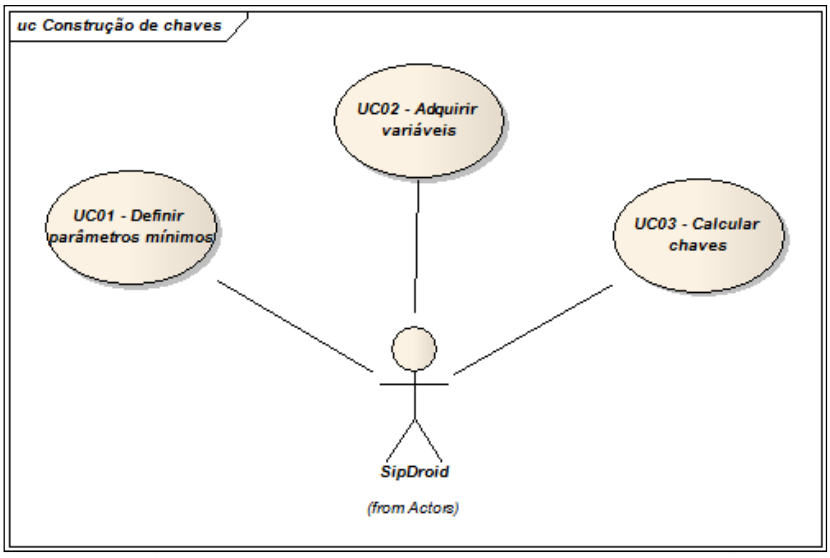


Figura 10 – Construção de chaves

Nos Quadros 13, 14 e 15, são apresentados os detalhamentos de cada caso de uso da fase de construção de chaves.

UC01 – Definir parâmetros mínimos	
Pré-condições	Estabelecer troca de pacotes com as outras partes da comunicação.
Cenário principal	1) Definir a existência de suporte a camada criptográfica 2) Enviar/Receber informações sobre versão da camada criptográfica suportada 3) Definição sobre as configurações mínimas suportadas por todos os participantes
Cenário alternativo	No passo 1, não existe suporte a camada criptográfica 1) Toda a comunicação ocorre normalmente, sem a interferência da camada criptográfica
Pós-condições	Participantes conhecem os parâmetros mínimos da comunicação

Quadro 13 – Detalhamento `UC01 – Definir parâmetros mínimos`

UC02 – Adquirir variáveis	
Pré-condições	Participante conhece os parâmetros mínimos da comunicação.
Cenário principal	1) Definir curva elíptica para a comunicação 2) Enviar/Receber a estrutura da equação da curva elíptica 3) Enviar/Receber os elementos comuns da curva elíptica definida
Cenário alternativo	No passo 2, os valores são recebidos incompletos 1) Requisitar re-envio dos dados
Pós-condições	Participantes conhecem os valores comuns da curva elíptica definida

Quadro 14 – Detalhamento `UC02 – Adquirir variáveis`

UC03 – Calcular chaves	
Pré-condições	Participante conhece os valores comuns da curva elíptica definida.
Cenário principal	1) Definir ponto `p` sobre a curva elíptica 2) Definir um valor escalar `a` 3) Efetuar a multiplicação escalar de `p` por `a`, gerando um ponto `q` 4) Definir o ponto `p` e `q` como chave pública e o escalar `a` como chave privada
Pós-condições	Chaves pública e privada geradas

Quadro 15 – Detalhamento `UC03 – Calcular chaves`

3.2.1.2 Negociação de chaves

A Figura 11 apresenta os casos de uso referentes a fase de negociação e troca das chaves públicas para a comunicação VoIP segura, tendo como único ator o próprio *softphone* SipDroid.

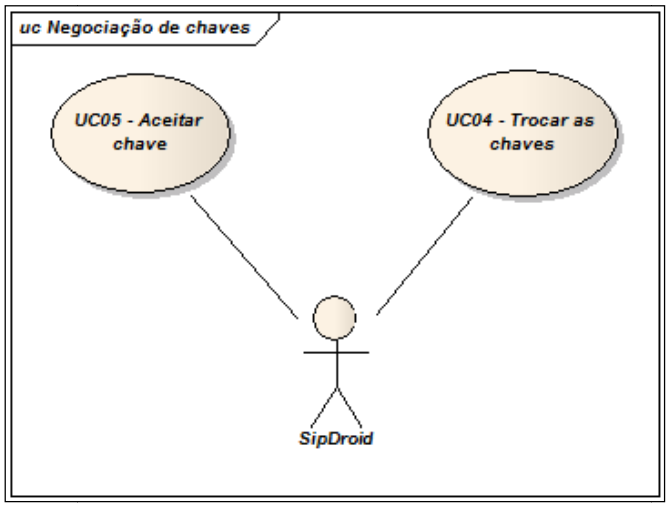


Figura 11 – Negociação de chaves

Nos Quadros 16 e 17, são apresentados os detalhamentos de cada caso de uso da fase de negociação de chaves.

UC04 – Trocar as chaves	
Pré-condições	Chaves pública-privada geradas.
Cenário principal	1) Enviar para cada participante a sua chave pública 2) Receber de cada participante a respectiva chave pública
Cenário alternativo	No passo 2, caso a chave chegue incompleta 1) Requisitar o reenvio da chave pública
Pós-condições	Participantes conhecem as chaves públicas os demais

Quadro 16 – Detalhamento `UC04 – Trocar as chaves`

UC05 – Aceitar chave	
Pré-condições	Chave pública de outro participante foi recebida.
Cenário principal	1) Verificar se cada ponto da chave pública pertence a curva elíptica definida
Cenário alternativo	No passo 1, caso não pertença 1) Requisitar o reenvio da chave pública
Pós-condições	Chave pública do participante é válida

Quadro 17 – Detalhamento `UC05 – Aceitar chave`

3.2.1.3 Utilização de chaves

A Figura 12 apresenta os casos de uso referentes a fase de utilização das chaves criptográficas da comunicação VoIP segura. Estes são os casos de uso que efetivamente permitem que os dados trafegados sejam sigilosos à terceiros.

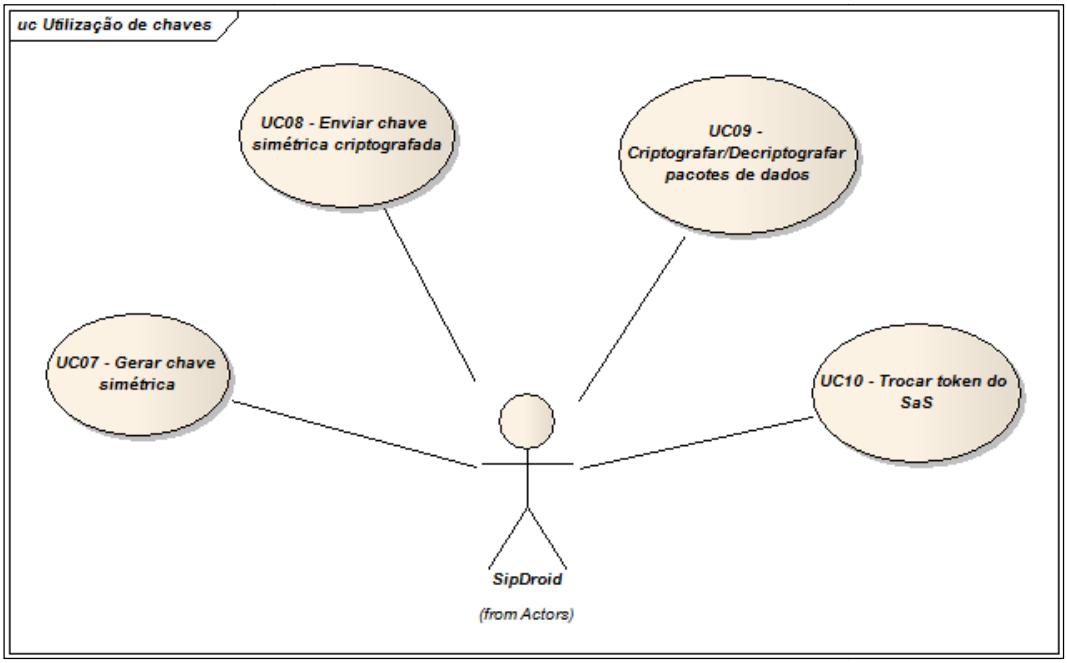


Figura 12 – Utilização de chaves

Nos Quadros 18, 19, 20 e 21, são apresentados os detalhamentos de cada caso de uso da fase de utilização de chaves.

UC07 – Gerar chave simétrica	
Pré-condições	Todas as chaves públicas foram efetivamente trocadas entre os participantes.
Cenário principal	1) Definir o algoritmo simétrico a ser utilizado 2) Selecionar um bom RND 3) Gerar uma chave simétrica a partir do RND
Cenário alternativo	No passo 1, caso o algoritmo não exista no ambiente 1) Enviar mensagem para cancelar toda a comunicação segura
Pós-condições	Chave simétrica para a comunicação gerada

Quadro 18 – Detalhamento `UC07 – Gerar chave simétrica`

UC08 – Enviar chave simétrica criptografada	
Pré-condições	Chave simétrica gerada
Cenário principal	Para cada participante: 1) Recuperar chave pública 2) Criptografar chave simétrica com chave pública 3) Enviar a chave simétrica criptografada
Cenário alternativo	No passo 3, caso participante esteja incomunicável 1) Efetuar até 10 tentativas de entrega. Caso ainda esteja incomunicável, enviar mensagem para cancelar a comunicação segura
Pós-condições	Chave simétrica compartilhada com todos os participantes

Quadro 19 – Detalhamento `UC08 – Enviar chave simétrica criptografada`

UC09 – Criptografar/Decriptografar pacotes de dados	
Pré-condições	Chave simétrica compartilhada entre todos os participantes
Cenário principal	1) Decriptografar o pacote recebido com a chave simétrica 2) Criptografar o pacote recebido com a chave simétrica
Pós-condições	Pacotes de dados criptografados/decriptografados

Quadro 20 – Detalhamento `UC09 – Criptografar/Decriptografar pacotes de dados`

UC10 – Trocar token do SAS	
Pré-condições	Já existe troca de pacotes de dados criptografados entre participantes
Cenário principal	1) Gerar um conjunto curto de caracteres 2) Enviar/Receber este conjunto curto de caracteres para todos os participantes
Cenário alternativo	No passo 2, caso o tamanho do conjunto seja diferente de 4 caracteres 1) Requisitar reenvio do conjunto de caracteres
Pós-condições	Conjunto de caracteres para autenticação (SAS) enviado a todos os participantes

Quadro 21 – Detalhamento `UC10 – Trocar token de SAS`

3.2.2 Arquitetura do Software

Nesta seção será apresentada a arquitetura do software desenvolvido, como também a

definição dos termos utilizados neste trabalho para melhor entendimento.

Por se tratar da alteração de um software previamente existente, a arquitetura do software desenvolvido foi herdada do *softphone* SipDroid, tendo somente adicionado elementos e dependências pertinentes a segurança de dados. A Figura 13 apresenta a arquitetura original do SipDroid.

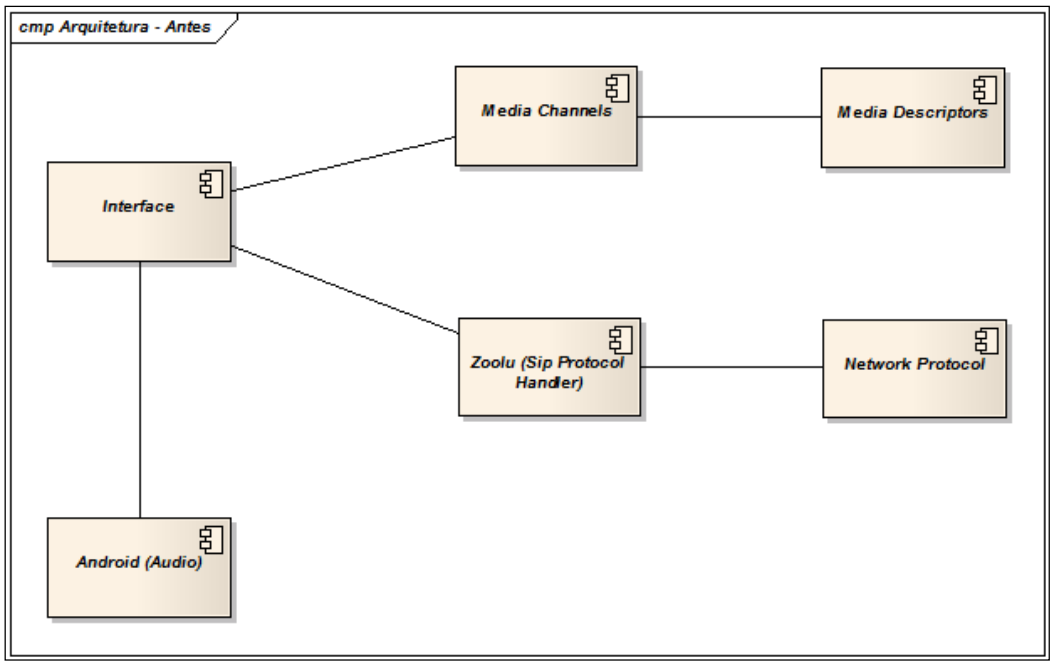


Figura 13 – Arquitetura SipDroid

A Figura 14 apresenta a arquitetura do SipDroid após as alterações que provêm segurança na comunicação.

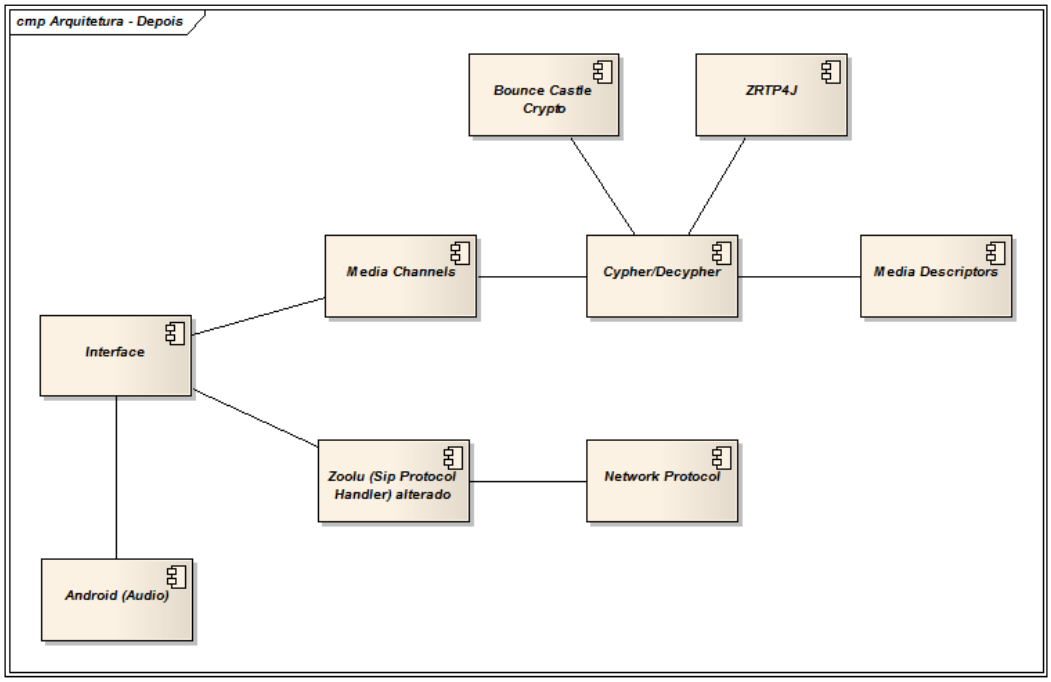


Figura 14 – Arquitetura alterada do SipDroid

Os principais componentes arquiteturais envolvidos são assim definidos:

a) interface: interface gráfica do *Softphone*, que possibilita a entrada e saída de dados, sejam eles textuais ou áudio-visuais;
b) Android (áudio): componente da plataforma Android responsável por processar o áudio recebido de qualquer aplicação e transferi-lo para o aparelho;
c) *media channels*: estruturas de dados que canalizam da uma das mídias envolvidas numa comunicação VoIP;
d) Zoolu: pilha SIP utilizada para o desenvolvimento do *Softphone*;
e) *network protocol*: estruturas de dados para a correta manipulação de cada tipo de protocolo de rede utilizado;
f) *media sescriptors*: estruturas de dados que descrevem cada uma das diferentes modulações de áudio e vídeo disponíveis;
g) *cypher/decypher*: componente que foi desenvolvido e que é o responsável por armazenar as cifras e efetuar a encriptação/decriptação dos dados trafegados;
h) *bouncy castle crypto*: componente responsável por fornecer os algoritmos e subsídios necessários para ambas criptografias, simétrica e assimétrica;
i) ZRTP4J: componente de terceiros que foi alterado, gerando uma versão 1.2, e que é o responsável por realizar a comunicação para o protocolo ZRTP.

3.2.3 Diagrama de classe

Nesta seção são apresentadas as classes utilizadas no desenvolvimento da camada criptográfica e a sua integração com o SipDroid. Estas classes são apresentadas de acordo com a arquitetura anteriormente apresentada.

As classes contidas no componente *Cypher/Decypher* contêm a lógica principal da camada de segurança desenvolvida. As classes contidas no componente Interface e *Media Channels* possuem alterações que possibilitam a utilização do componente *Cypher/Decypher*.

A Figura 15 exibe as classes mais relevantes dos componentes Interface e *Media Channels*.

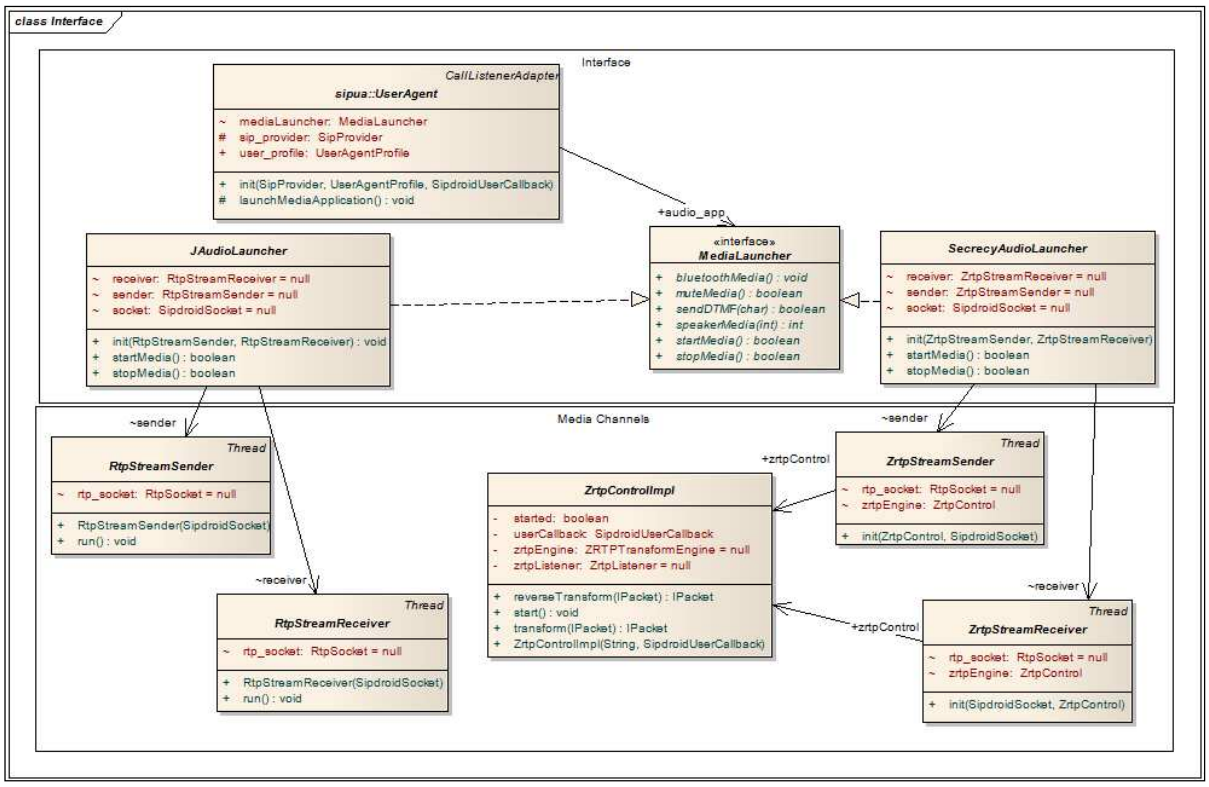


Figura 15 – Classes dos componentes Interface e *Media Channels*

Segue o detalhamento das classes do componente Interface e *Media Channels*:

a) `UserAgent`: classe original do componente Interface, que representa a interface gráfica com o usuário;

b) `MediaLaucher`: *interface* do componente Interface desenvolvida para definir os métodos básicos para o gerenciamento dos *streams*;

c) `JAudioLauncher`: classe do componente Interface responsável por gerenciar os *streams*, tanto de saída quanto de entrada, e por definir os parâmetros para a captura e execução do áudio;

d) `RtpStreamReceiver`: classe inalterada do componente *Media Channels* responsável por receber os dados não-criptografados da comunicação e delegar a execução do som à plataforma Android;

e) `RtpStreamSender`: classe inalterada do componente *Media Channels* responsável capturar o áudio da plataforma Android e enviá-los aos participantes da comunicação não-segura;

f) `SecrecyAudioLauncher`: classe do componente Interface desenvolvida para gerenciar os *streams* quando estes trafegam os dados de forma criptografada, além de definir os parâmetros básicos para a captura e execução do áudio;

g) `ZrtpStreamReceiver`: classe do componente *Media Channels* desenvolvida para

receber os dados criptografados da comunicação, chamar a rotina de decriptação do componente *Cypher/Decypher* e delegar a execução do áudio para o componente Android;

h) `ZrtpStreamSender`: classe do componente *Media Channels* desenvolvida para capturar os dados de áudio do componente Android, chamar a rotina de encriptação do componente *Cypher/Decypher* e enviá-los aos participantes da comunicação;

i) `ZrtpControlImpl`: classe do componente *Media Channels* que foi desenvolvida para se comunicar com o componente *Cypher/Decypher*, através de um objeto do tipo `ZrtpTransformEngine`.

As Figuras 16.1 e 16.2 exibem as classes do componente ZRTP4J (versão 1.2) e algumas classes de componentes externos, usadas para comunicação com os mesmos.

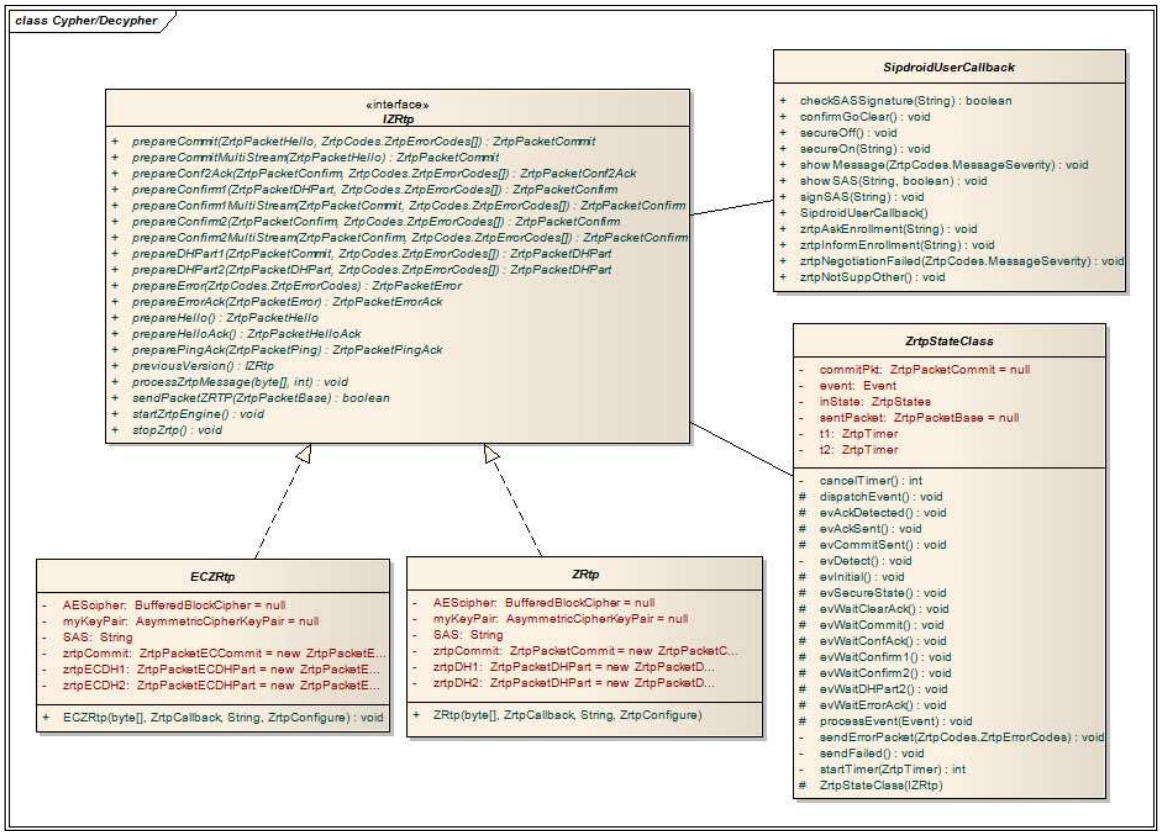


Figura 16.1 – Classes do componente ZRTP4J (versão 1.2)

Segue o detalhamento das classes da Figura 16.1:

a) `IZRtp`: interface desenvolvida para abstração do tipo de criptografia de chaves utilizadas. A função de uma classe que estenda `IZRtp` é gerenciar a troca de pacotes do protocolo ZRTP e criptografar os dados. É somente acessada por um objeto do tipo `ZrtpTransformEngine`, que é o responsável pela ligação entre os componentes ZRTP4J (versão 1.2) e *Media Channels*;

b) `ZRtp`: classe do projeto ZRTP4J (versão 1.1). Permite a geração e troca de chaves do protocolo ZRTP utilizando o problema de logaritmo discreto;

c) `ECZRtp`: classe desenvolvida para o ZRTP4J (versão 1.2) que permite a geração e troca de chaves do protocolo ZRTP utilizando o problema de logaritmo discreto de curvas elípticas;

d) `SipdroidUserCallback`: classe do componente Interface que foi desenvolvida para fornecer rotinas de *callbacks* das principais rotinas do protocol ZRTP;

e) `ZrtpStateClass`: classe originalmente do projeto ZRTP4J (versão 1.1), mas que teve de ser alterada para o suporte a criptografia de chaves com curvas elípticas. Responsável por processar cada evento e pacote recebido das demais partes da comunicação onde se tenta estabelecer o protocolo ZRTP;

A Figura 16.2 exibe as classes que representam as mensagens que são trocadas entre os participantes de uma comunicação pelo protocolo ZRTP.

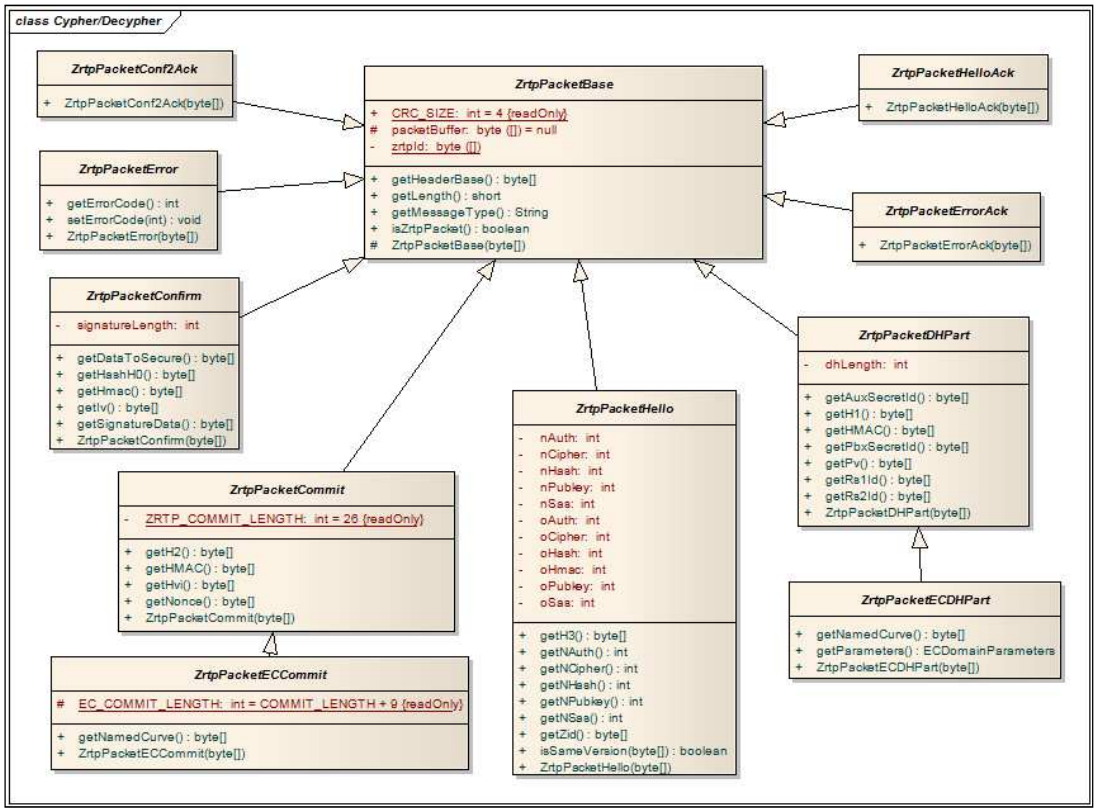


Figura 16.2 – Classes do componente ZRTP4J (versão 1.2)

Segue o detalhamento das classes de mensagens ZRTP, mostradas na Figura 16.2:

a) `ZrtpPacketBase`: classe básica original do projeto ZRTP4J que contém as informações em comum entre todos os tipos de pacotes trocados;

b) `ZrtpPacketHello`: classe original do projeto ZRTP4J que representa o primeiro pacote trocado. Este pacote contém todas as informações necessárias para a

criação das chaves, *salt* e da SAS e também, qual o algoritmo simétrico a ser utilizado;

c) `ZrtpPackerHelloAck`: classe original do projeto ZRTP4J que representa a mensagem de recepção com sucesso de uma mensagem do tipo `ZrtpPacketHello`;
d) `ZrtpPacketCommit`: classe original do projeto ZRTP4J que representa a mensagem de aceitação dos parâmetros presentes no pacote `ZrtpPacketHello`;
e) `ZrtpPacketECCommit`: classe desenvolvida para o ZRTP4J (versão 1.2) que especializa a classe `ZrtpPacketCommit` e adiciona as informações da curva elíptica escolhida para a geração das chaves;
f) `ZrtpPacketDHPart`: classe original do projeto ZRTP4J que representa uma parte da chave Diffie-Helmann da comunicação;
g) `ZrtpPacketECDHPart`: classe desenvolvida para o ZRTP4J (versão 1.2) que especializa a classe `ZrtpPacketDHPart` e adiciona as informações da curva elíptica escolhida para a geração das chaves;
h) `ZrtpPacketConfirm`: classe original do projeto ZRTP4J que representa a confirmação do recebimento da parte da chave Diffie-Helmann, e que também contém alguns parâmetros para a verificação das partes da comunicação;
i) `ZrtpPacketConf2Ack`: classe original do projeto ZRTP4J que representa a mensagem de recepção com sucesso de uma mensagem do tipo `ZrtpPacketConfirm`;
j) `ZrtpPacketError`: classe original do projeto ZRTP4J que representa uma mensagem de erro, onde deve existir um código do erro ocorrido;
k) `ZrtpPacketErrorAck`: classe original do projeto ZRTP4J que representa a mensagem de recepção com sucesso de uma mensagem do tipo `ZrtpPacketError`.

3.2.4 Diagrama de sequência

Esta seção apresenta o diagrama de sequência que representa o conjunto de passos que o programa executa para realizar determinada tarefa, com base nas ações do usuário.

A Figura 17 exibe o diagrama de sequência da criação das chaves, tanto pública

quando privada, com a criação de um PRNG específico para o protocolo ZRTP, o `ZRTPFortuna`, que utiliza o `FortunaGenerator`, um PRNG do projeto GNU que foi adaptado à linguagem Java.

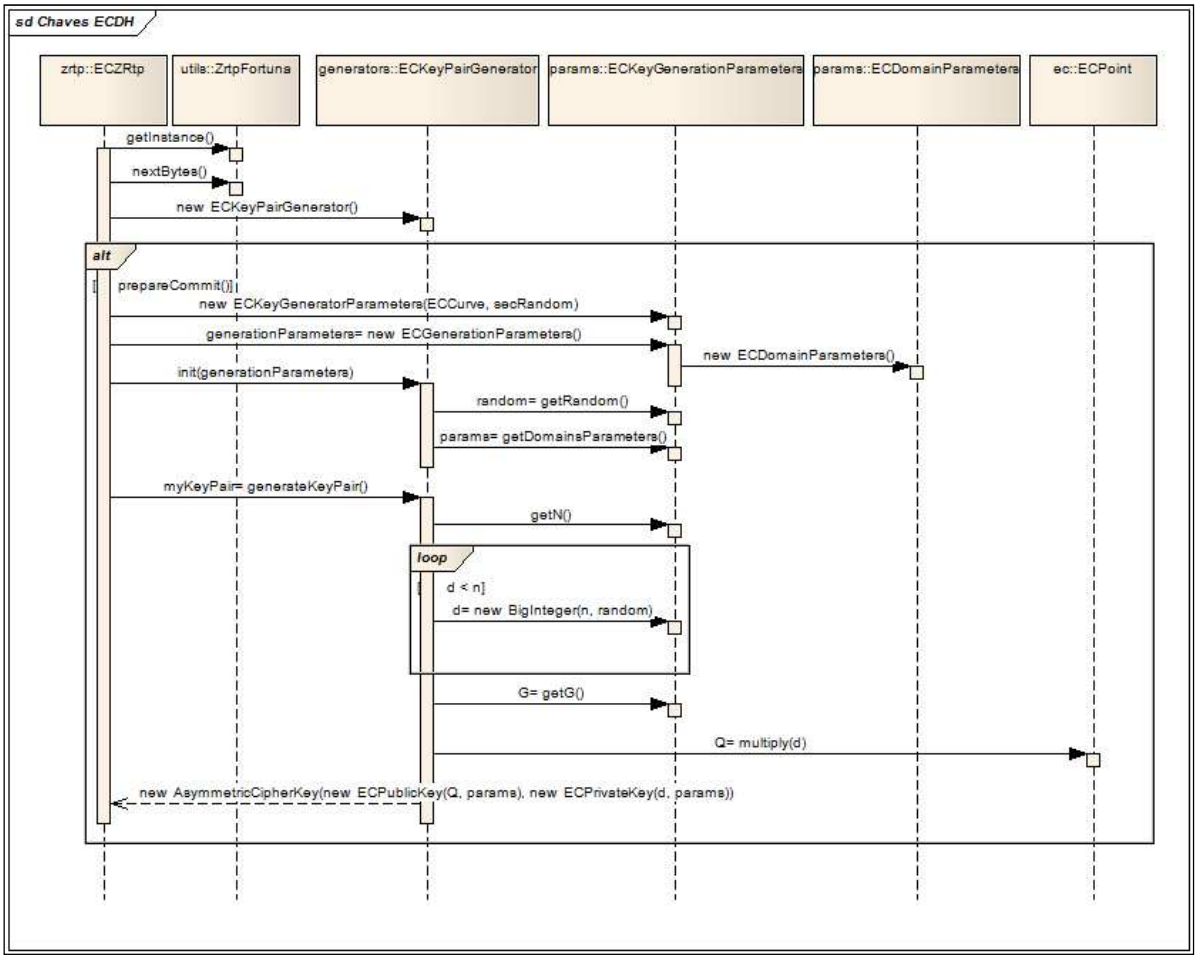


Figura 17 – Diagrama de sequência da criação das chaves

3.3 IMPLEMENTAÇÃO

Nesta seção serão apresentadas as técnicas e ferramentas utilizadas, o desenvolvimento da camada criptográfica e a operacionalidade do *softphone*.

3.3.1 Técnicas e ferramentas utilizadas

A camada criptográfica foi desenvolvida na linguagem Java, para a plataforma Android, seguindo o paradigma de orientação à objetos. Todo o desenvolvimento foi

realizado com o ambiente de desenvolvimento Eclipse. Para a realização dos testes, foi utilizado um aparelho Nexus One, *smartphone* focado para o desenvolvimento para a plataforma Android, e foi usado como servidor VoIP a ferramenta 3CX *Phone System*, da empresa 3CX Ltda.

3.3.2 ZRTP4J

O ZRTP4J é uma biblioteca que permite a comunicação pelo protocolo ZRTP na plataforma Java padrão. É uma biblioteca do projeto GNU, que está sobre a licença GPL, versão três. Esta licença exige que o código-fonte da biblioteca seja livre e que esteja disponível a todos.

A biblioteca foi desenvolvida para a plataforma Java padrão, então houve a necessidade de utilizar o seu código-fonte e compilá-los para a plataforma Android. Esta também é a única implementação compatível do protocolo ZRTP para a plataforma Java/Android. Todas as outras implementações encontradas possuem somente as suas versões binárias e são exclusivas de outras linguagens.

Esta é uma biblioteca bem completa e autônoma, possuindo uma utilização simples, sendo necessário somente ter o conhecimento sobre como integrar o seu único *plugin*, para a biblioteca *Java Media Framework* (JMF), com a plataforma desejada, nesse caso a camada de áudio da plataforma Android.

Para o desenvolvimento do presente trabalho, foi necessário o desenvolvido de uma extensão do projeto ZRTP4J, para que este provenha suporte a criptografia de chaves através de curvas elípticas. Esta extensão foi denominada como versão 1.2, visto que mantêm a compatibilidade com a versão 1.1 e foram adicionadas novas mensagens para a negociação e construção das chaves assimétricas.

No Quadro 14 é exibido o código utilizado no protótipo que faz a inicialização da biblioteca ZRTP4J (versão 1.2), e também o código que faz uso da mesma, para a cifragem dos dados.

```
private void init(boolean do_sync, Codecs.Map payload_type,
                  long frame_rate, int frame_size, SipdroidSocket
            src_socket, String dest_addr, int dest_port,
      String zidFilename, ZrtpControl zrtpControl) {
      ...
      rtp_socket = new RtpSocket(src_socket, InetAddress
                  .getByName(dest_addr), dest_port);

      zrtpEngine = zrtpControl;
      zrtpEngine.init(rtp_socket);
      zrtpEngine.start();
}

public void run() {
      ...
      duration = (int) (time - dttime);
      dt_packet.setSequenceNumber(seqn);
      dt_packet.setTimestamp(dttime);
      dtmfbuf[12] = rtpEventMap.get(dtmf.charAt(0));
      dtmfbuf[13] = (byte) 0x8a;
      dtmfbuf[14] = (byte) (duration >> 8);
      dtmfbuf[15] = (byte) duration;
      try {
            rtp_socket.send(zrtpEngine.transform(dt_packet));
      } catch (IOException e1) {
      }
      ...
}
```

Quadro 14 – Código de inicialização e utilização do ZRTP4J, versão 1.2

As Figuras 18.1 e 18.2 exibem o diagrama de atividade que representa todas as etapas e trocas de mensagens exigidas pelo protocolo ZRTP, em sua versão 1.2, tanto em modo *Initiator* quanto em modo *Responder*.

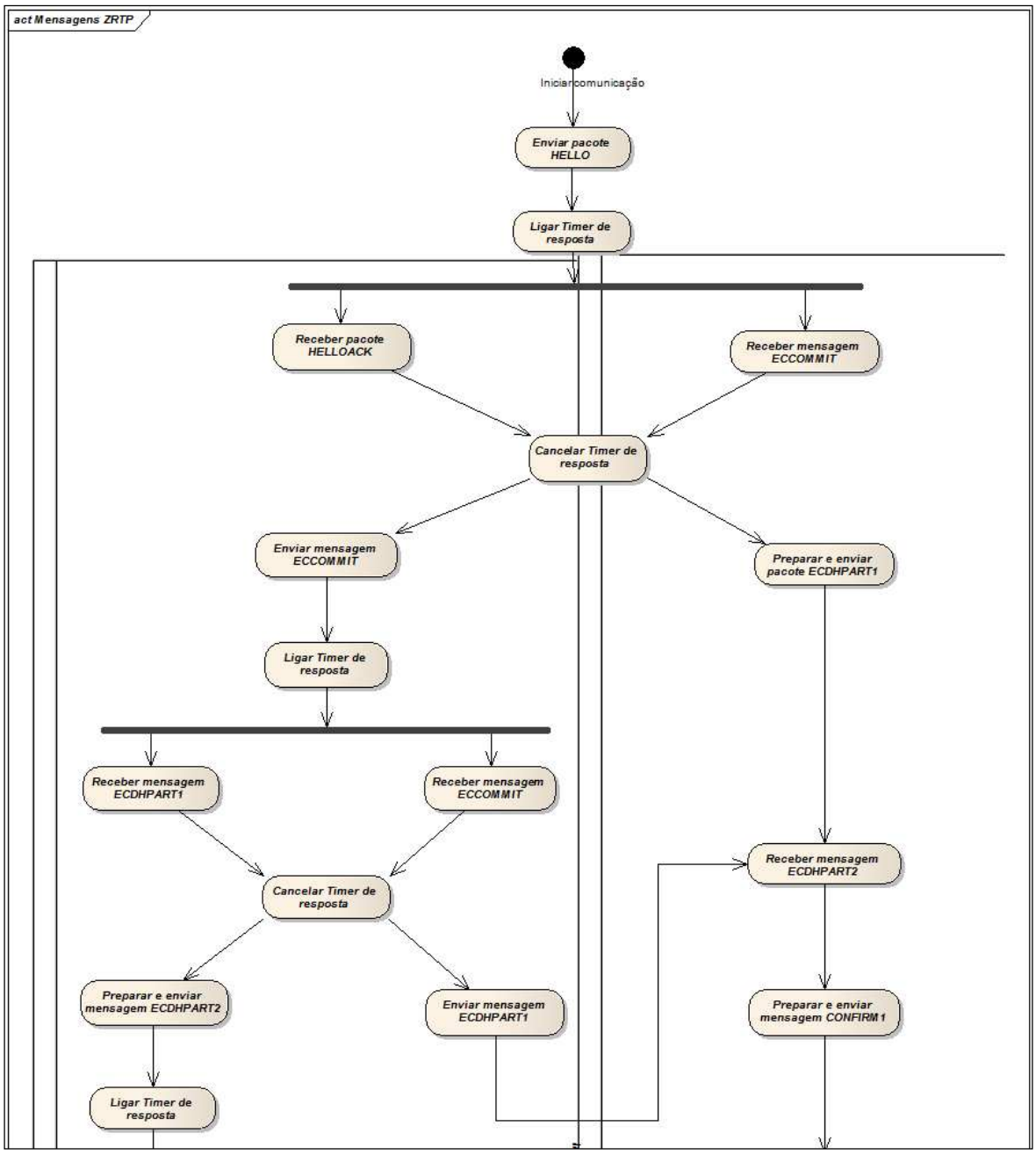


Figura 18.1 – Diagrama de atividade das mensagens ZRTP

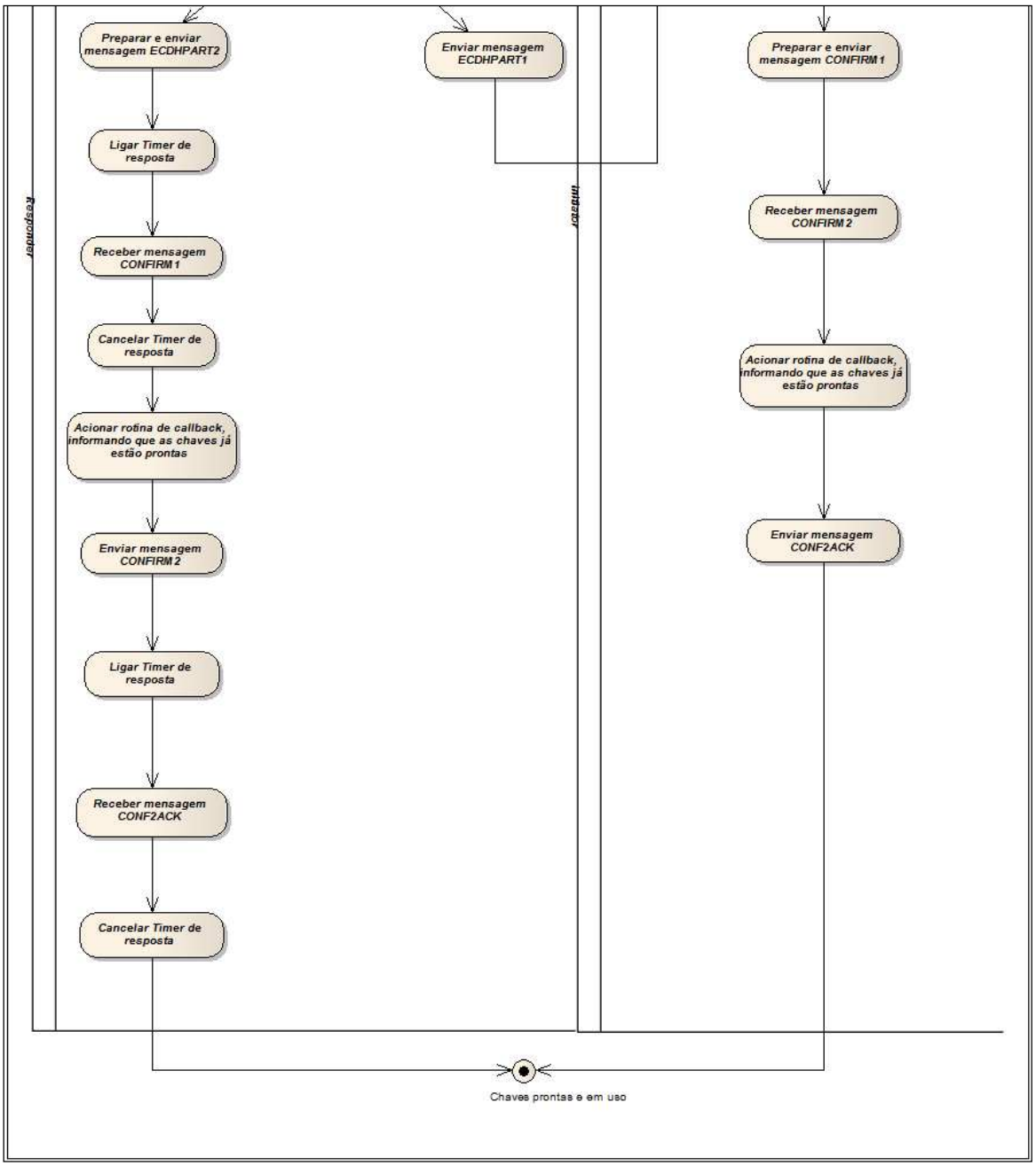


Figura 18.2 – Diagrama de atividade das mensagens ZRTP

3.3.3 Bouncy *Castle Crypto*

A biblioteca Bouncy *Castle Crypto* é uma API para criptografia, que visa proporcionar implementações leves e simples dos principais algoritmos abertos existentes. Ele possui implementação em Java e em C#. A sua versão Java é compatível com as APIs *Java Cryptography Extension* e *Java Cryptography Architecture*. Ela está disponível sob a licença MIT X11, permitindo que qualquer pessoa possa ler e utilizar o mesmo de forma livre de

qualquer obrigação legal.

Por esta também ser uma biblioteca desenvolvida para a plataforma Java, foi necessário a recompilação do seu código-fonte para a plataforma Android e a adaptação de algumas poucas classes, para o seu correto funcionamento.

A biblioteca Bouncy *Castle Crypto* é uma biblioteca referência em algoritmos criptográficos, pois além da grande gama de algoritmos suportados, possui uma API de fácil assimilação e desenvolvimento.

O Quadro 15 exibe o código desenvolvido pelo protótipo que é responsável pela geração da chave criptográfica, utilizando curvas elípticas.

```
String nameOfCurve = SECNamedCurves.SECP160R2;
dhKeyPairGen.init(new ECKeyGenerationParameters(
          ZrtpUtils.getDomainParameters(nameOfCurve),
          new SecureRandom(secRand.getFortuna().getSeedStatus())));
int pubKeySize = 160;
myKeyPair = dhKeyPairGen.generateKeyPair();
```

Quadro 15 – Código que efetua a multiplicação por escalar sobre um ponto

3.3.4 Desenvolvimento da camada criptográfica

A partir do momento da primeira troca de mensagens SIP, é feita a detecção de qual versão do protocolo ZRTP está sendo utilizada por ambas as partes da comunicação através do atributo `zrtp-hash`, que fica no corpo da mensagem SIP. Esse atributo é padronizado pela especificação do protocolo ZRTP e deve ser declarado no pacote de `INVITE` da comunicação SIP. Existem dois níveis de criptografia suportados: a versão 1.1, que utiliza o algoritmo de Diffie-Helmann original, e a versão 1.2, que utiliza o algoritmo de Diffie-Helmann com curvas elípticas. Estes dois métodos de troca de chaves são incompatíveis, e foi esta incompatibilidade que levou a criação de uma versão 1.2 para o protocolo ZRTP.

Após a distinção, é iniciada a troca de pacotes do protocolo ZRTP, com o fluxo anteriormente descrito nas Figuras 20.1 e 20.2. Toda esta troca de mensagens ocorre paralelamente a troca de mensagens SIP e a troca de pacotes de dados. Ou seja, até que o protocolo ZRTP complete a sua troca de mensagem, todo o conteúdo trafegado fica desprotegido.

O Quadro 17 exibe o código responsável por chamar a rotina de criptografia dos dados.

```
public IPacket transform(IPacket pkt) {
    byte[] buffer = pkt.getPacket();
    int offset = 0;
    if ((buffer[offset] & 0x10) == 0x10)  {
        return pkt;
    }
    /*
     * ZRTP necessita do SSRC da mensagem a ser enviada.
     */
    if (enableZrtp && ownSSRC == 0) {
        ownSSRC = (int) pkt.getSscr();
    }
    /*
     * Se o SRTP está ativo então srtpTransformer existe e deve ser
utilizado
     */
    sendPacketCount++;
    if (srtpOutTransformer == null) {
        return pkt;
    }
    return srtpOutTransformer.transform(pkt);
}
```

Quadro 17 – Código que chama a rotina de criptografia

O Quadro 18 exibe a operação inversa, a decriptografia dos dados. Esta funcionalidade tem algumas particularidades, pois nem sempre o protocolo ZRTP completou o seu processo de estabelecimento das chaves. Nestes casos, alguns pacotes do protocolo devem ser emulados, para que o processo se complete, para só então ocorrer a troca de dados criptografados.

```
public IPacket reverseTransform(IPacket pkt) {
    // Garante que o protocolo ZRTP está iniciado
    if (!started && enableZrtp && ownSSRC != 0) {
        startZrtp();
    }
    /*
     * Verifica se o pacote é um pacote ZRTP. Caso negativo,
     * trata ele como um pacote RTP normal.
     */
    byte[] buffer = pkt.getPacket();
    int offset = 0;
    if ((buffer[offset] & 0x10) != 0x10) {
        if (srtpInTransformer == null) {
            return pkt;
        }
        pkt = srtpInTransformer.reverseTransform(pkt);
        // Se o pacote é válido (não-nulo) e o ZRTP já está iniciado, mas
ainda não está completo, emular um pacote CONFIRM2ACK. Ver a especificação
ZRTP, capítulo 5.6
        if (pkt != null && started) {
            if(zrtpEngine.inState(ZrtpStateClass.ZrtpStates.WaitConfAck)) {
                  zrtpEngine.conf2AckSecure();
            }
        }
        return pkt;
    }

    /*
     * Se o ZRTP está ativo, processa o pacote. Caso ocorra qualquer
problema, deve ser retornado null, pois um pacote ZRTP nunca pode chegar à
aplicação
     */
    if (enableZrtp && started) {
        ZrtpRawPacket zPkt = new ZrtpRawPacket(pkt);
        if (!zPkt.checkCrc()) {
            userCallback.showMessage(ZrtpCodes.MessageSeverity.Warning,
                  EnumSet.of(ZrtpCodes.WarningCodes.WarningCRCmismatch));
            return null;
        }
        // Garante que é realmente um pacote ZRTP
        if (!zPkt.hasMagic()) {
            return null;
        }
        byte[] extHeader = zPkt.getMessagePart();
        zrtpEngine.processZrtpMessage(extHeader, zPkt.getSSRC());
    }
    return null;
}
```

Quadro 18 – Código que chama a rotina de decriptografia

A chamada a `srtpInTransformer`, em síntese, somente encapsula uma chamada ao componente *Bouncy Castle Crypto*, que possui a implementação do algoritmo simétrico AES, trabalhando com dois modos distintos de criptografia AES: *Counter Mode* e F8 *Mode*.

Quando o protocolo ZRTP completa a troca de chaves, e esta ocorre sem problemas, é necessário que todos os participantes da comunicação confirmem a SAS, para inibir um eventual ataque do tipo *Man-in-the-Middle*, onde existe um interceptador na comunicação.

Este interceptador alteraria o estado da conexão, gerando para cada participante da comunicação um valor da SAS diferente.

O Quadro 19.1 apresenta o método da classe `SipdroidUserCallback` que notifica o usuário, através de um ícone de um cadeado aberto, que a conexão entre as partes não é segura.

```
public void secureOff() {
      String ns = Context.NOTIFICATION_SERVICE;
      NotificationManager mNotificationManager = (NotificationManager)
uiContext.getSystemService(ns);
      int icon = R.drawable.ic_cadeado_aberto;
      CharSequence tickerText = "Unsecure call";
      long when = System.currentTimeMillis();
      Notification notification = new Notification(icon, tickerText, when);
      Context context = uiContext;
      CharSequence contentTitle = "Secure Off";
      CharSequence contentText = "ZRTP connect isn't secure";
      PendingIntent intent = PendingIntent.getActivity(uiContext, 0, new
Intent(), 0);
      notification.setLatestEventInfo(context, contentTitle, contentText,
intent);
      mNotificationManager.notify(SECURE_OFF, notification);
      super.secureOff();
}
```

Quadro 19.1 – Método que notifica que a conexão não é segura

O Quadro 19.2 apresenta o método que notifica o usuário, através de um ícone de um cadeado fechado, que a conexão entre as partes é segura.

```
public void secureOn(String cipher) {
      String ns = Context.NOTIFICATION_SERVICE;
      NotificationManager mNotificationManager = (NotificationManager)
uiContext.getSystemService(ns);
      int icon = R.drawable.ic_cadeado_fechado;
      CharSequence tickerText = "Secure call";
      long when = System.currentTimeMillis();
      Notification notification = new Notification(icon, tickerText, when);
      PendingIntent activity = PendingIntent.getActivity(uiContext, 0, new
Intent(), 0);
      String contentText = "ZRTP connect is secure";
      String contentTitle = "Secure on";
      notification.setLatestEventInfo(uiContext, contentTitle, contentText,
activity);
      mNotificationManager.notify(SECURE_ON, notification);
      super.secureOn(cipher);
}
```

Quadro 19.2 – Método que notifica que a conexão é segura

O Quadro 19.3 apresenta o método que notifica o usuário, através de uma notificação na barra de *status* da plataforma Android, qual é a SAS utilizada durante aquela sessão VoIP.

```
public void showSAS(String sas, boolean verified) {
      String ns = Context.NOTIFICATION_SERVICE;
      NotificationManager mNotificationManager = (NotificationManager)
uiContext.getSystemService(ns);
      int icon = R.drawable.ic_cadeado_fechado;
      CharSequence tickerText = "Secure call";
      long when = System.currentTimeMillis();
      Notification notification = new Notification(icon, tickerText, when);
      RemoteViews contentView = new RemoteViews(uiContext.getPackageName(),
R.layout.zrtp_sas);
      contentView.setImageViewResource(R.id.sas_image, R.drawable.icon32);
      String msg=uiContext.getResources().getString(R.string.zrtp_sas,
sas);
      contentView.setTextViewText(R.id.sas_text, msg);
      notification.contentView = contentView;
      PendingIntent intent = PendingIntent.getActivity(uiContext, 0, new
Intent(), 0);
      notification.contentIntent = intent;
      mNotificationManager.notify(SECURE_ON, notification);
      super.showSAS(sas, verified);
}
```

Quadro 19.3 – Método que notifica a SAS definida para a comunicação

3.3.5 Operacionalidade

Nesta seção será apresentada a operacionalidade da camada criptográfica desenvolvida e também a efetiva criptografia dos dados. Inicialmente, o SipDroid exibe um campo para a entrada de dados sobre o destinatário da comunicação VOIP, conforme exibido na Figura 19.

Figura 19 – Tela inicial do SipDroid

O servidor que estava configurado para esta execução do SipDroid (192.168.0.100) possui somente dois participantes cadastrados: 11 e 12. Para realizar uma ligação, bastaria

digitar o endereço desejado e teclar o ícone de telefone. A Figura 20 exibe um início de uma ligação para o participante 11.

Figura 20 – Chamada ao participante 11

Neste momento, existe a negociação da conexão entre os participantes da comunicação, e também são negociados os parâmetros básicos da chamada, incluindo aqui a presença ou não de criptografia, e sua respectiva versão. A Figura 21 exibe o momento em que o outro participante atende a ligação.

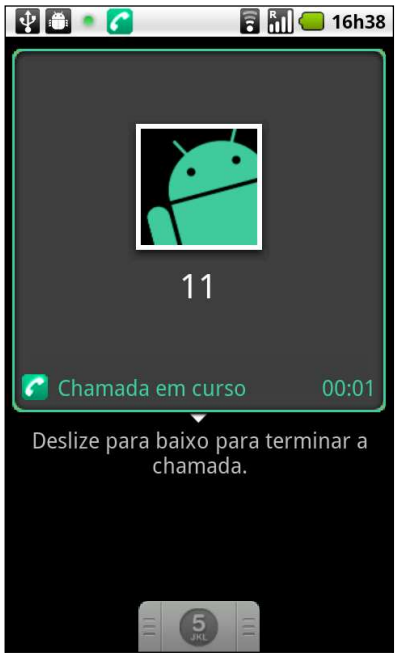


Figura 21 – Chamada já em curso

Neste instante, a comunicação do protocolo SIP cessa, e se inicia a troca de pacotes de dados pelo protocolo RTP. Caso ambos os lados da comunicação suportem uma mesma

versão, irá se iniciar a negociação do protocolo ZRTP, para a troca das chaves simétricas. Enquanto esta negociação não se concretiza, a comunicação entre as partes se realiza normalmente, sem qualquer tipo de segurança. Na Figura 22 é exibida a mensagem de conclusão da negociação das chaves.

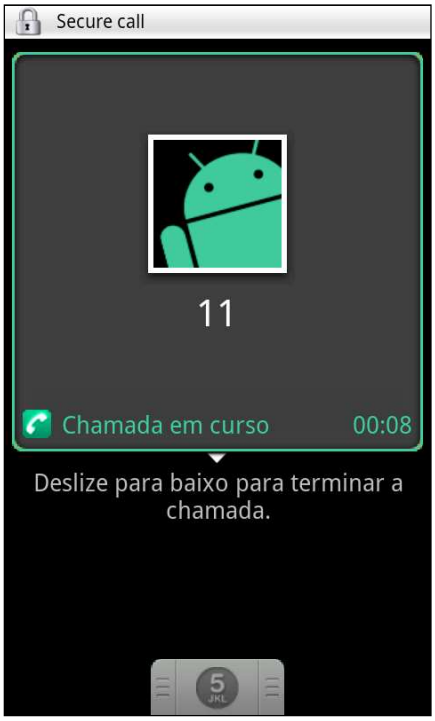


Figura 22 – Conclusão da negociação das chaves

A partir deste instante, toda a comunicação de dados entre as partes ocorre de forma segura, com a criptografia dos pacotes de dados sendo realizada com o algoritmo AES e a sua chave já devidamente trocada entre as partes. A Figura 23 exibe a continuidade dessa comunicação segura entre as partes, com um ícone de um cadeado fechado no canto superior esquerdo, representando que a comunicação ocorreu perfeitamente, e que as partes devem também verificar a sua SAS.

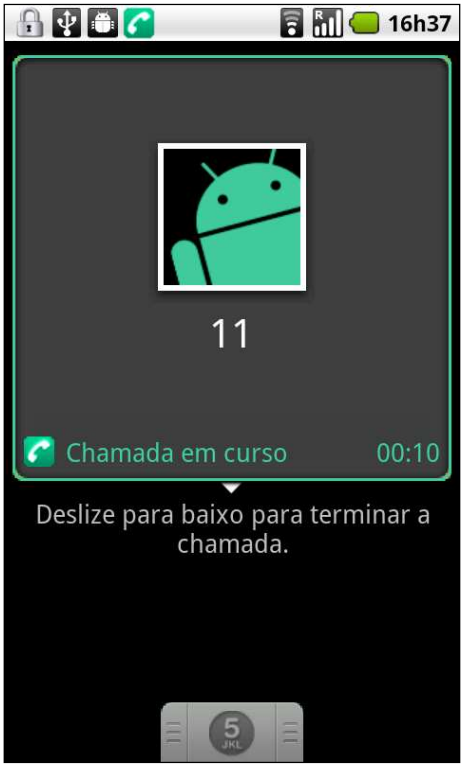


Figura 23 – Comunicação segura

A Figura 24 exibe a mensagem que contém a SAS da comunicação segura entre as partes. Para minimizar a possibilidade de um ataque do tipo *Man-in-the-Middle*, todas as partes devem comunicar aos outros participantes qual é a mensagem SAS que aparece na sua chamada. Caso sejam distintas, significa que há um ataque do tipo *Man-in-the-Middle* ocorrendo.

Figura 24 – Exibição da mensagem SAS

Com todas as partes reconhecendo a SAS como sendo a mesma, os participantes possuem maior confiança que a sua comunicação é segura, e que toda e qualquer mensagem trafegada por ela é indistinguível de dados aleatórios.

Como software para a análise dos dados criptografados, foi utilizada a ferramenta Wireshark, disponível em WIRESHARK (2010). Para uma análise mais correta, é necessário que a fonte de áudio seja constante, para haver uma clara distinção entre um áudio criptografado e um áudio não-criptografado. Para isso, foi utilizado comunicador sem um dispositivo de microfone, o que garante que todo o áudio emitido por ele seja idêntico em toda e qualquer chamada.

A Figura 25 apresenta uma comunicação SIP convencional, sem nenhum tipo de criptografia.

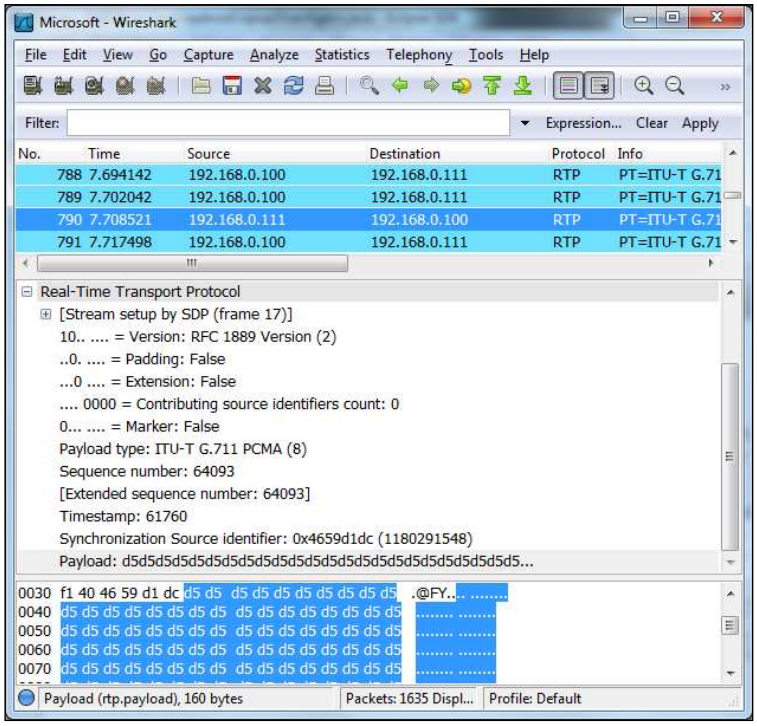


Figura 25 – Comunicação sem criptografia

Já a Figura 26 apresenta a mesma comunicação, porém com a camada criptográfica do SipDroid ativada e funcional.

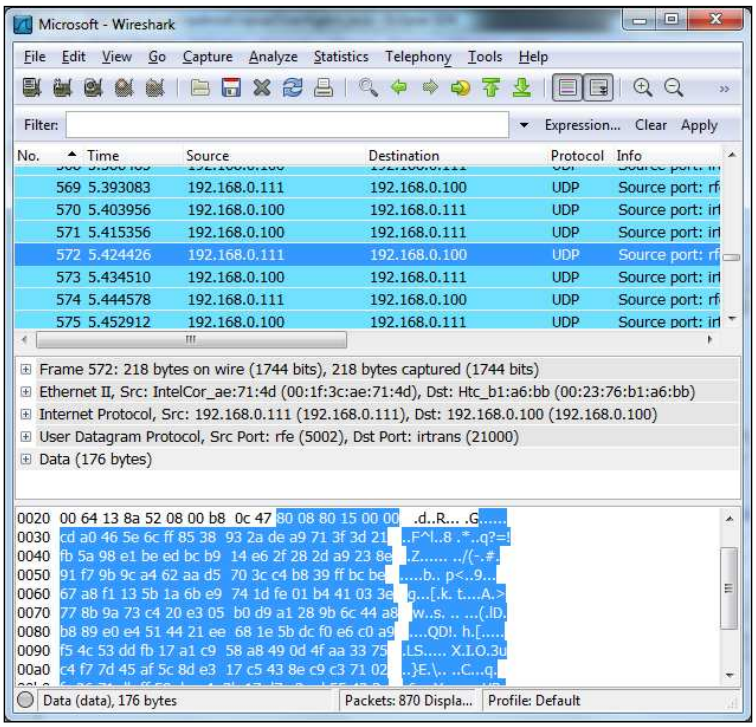


Figura 26 – Comunicação criptografada

3.4 RESULTADOS E DISCUSSÃO

O foco deste trabalho é fornecedor segurança, através de criptografia, à comunicação VoIP. Tendo este foco em mente, o presente trabalho possui as mesmas características do SIP *Communicator*. Tanto o SIP *Communicator* quanto o presente trabalho proporcionam a criptografia através do algoritmo AES, ambos utilizam a biblioteca ZRTP4J, e ambos utilizam o algoritmo de Diffie-Helmann para a troca das chaves simétricas. Existem dois pontos que divergem entre ambos: o presente trabalho foi feito para a plataforma Android, enquanto que o SIP *Communicator* foi desenvolvido para a plataforma Java, para executar em computadores convencionais; e o presente trabalho pode realizar a troca de chaves tanto com o algoritmo de Diffie-Helmann puro, utilizando o problema de logaritmo discreto para a geração das chaves, como também pode utilizar o Diffie-Helmann com curvas elípticas, que utiliza o problema de logaritmo discreto de curvas elípticas para a geração das chaves.

Quando comparado ao trabalho de Bazotti (2007), o presente trabalho proporcionou muito mais segurança e confiabilidade no tráfego dos dados. A escolha de Bazotti pela plataforma Java ME o limitou a um conjunto de APIs que quase não evoluiu nos últimos 3 anos, desde a data de publicação do trabalho. As dificuldades técnicas sofridas por Bazotti, é

possível concluir, foram culpa das poucas exigências que a plataforma Java ME faz dos aparelhos onde esta está instalada. A plataforma Android possui uma gama maior de exigências para que um aparelho seja aprovado para a utilização da mesma. Com isso, a integração entre o hardware e o software é muito mais suave e precisa, minguando os problemas e frustrações enfrentados, tanto por desenvolvedores quanto por usuários. Bazotti fez uma implementação de um protocolo semelhante ao SIP, com isso ele perdeu a capacidade de interoperar com outros comunicadores, enquanto que no trabalho presente, todo e qualquer comunicador SIP que implemente o protocolo ZRTP poderá se comunicar de forma segura com o mesmo.

A execução do SipDroid, em comunicações seguras, manteve a fluidez necessária para uma comunicação telefônica, não adicionando *delays* a recepção ou ao envio dos dados. A geração e a visualização da SAS ocorrem conforme a especificação do protocolo ZRTP, garantindo a intercomunicação com outros comunicadores VoIP que implementam o ZRTP. Com isso, cada um dos objetivos propostos para o trabalho pôde ser concluído com excelência.

4 CONCLUSÕES

A segurança das informações trafegadas é um assunto de extrema importância, desde a época dos reis gregos e dos faraós egípcios, que iniciaram a utilização de códigos secretos para a troca de mensagens sobre guerras e acordos diplomáticos entre nações. Da mesma forma, nos tempos atuais é necessária a troca de informações sigilosas, sejam entre as nações, entre empresas ou mesmo entre pessoas. O atual estado da criptografia permite que essa troca de informações ocorra com um bom nível de segurança, graças às técnicas matemáticas desenvolvidas para tais atividades. Hoje em dia não se usa mais a simples troca de símbolos, técnica primária, desenvolvida pelos egípcios, mas sim complexos modelos matemáticos, o que dificulta em muito a decifragem das mensagens.

As técnicas de trocas de chaves criptográficas que não exijam a presença de um terceiro confiável asseguram que somente os verdadeiros participantes da comunicação venham a ter conhecimento e poder sobre o conteúdo por ali trafegado. O algoritmo de Diffie-Helmann assegura isto através de uma troca de simples mensagens, contendo alguns parâmetros comuns entre as chaves criptográficas que, mesmo que um elemento não-participante roube os dados, esse não será capaz de reconstruir a mensagem, ou somente sendo capaz após um tempo de computação suficientemente grande a ponto de a informação não ter mais utilidade prática, graças ao problema matemático utilizado.

O trabalho exposto se propunha a criar uma comunicação VoIP, através do protocolo SIP, segura e que fosse funcional e prática para um usuário comum. É possível concluir que o objetivo foi cumprido, e até estendido, na medida em que foram implementados duas maneiras distintas de efetuar a troca das chaves criptográficas: algoritmo de Diffie-Helmann puro e algoritmo de Diffie-Helmann com chaves elípticas. O segundo garante, teoricamente, uma maior segurança quando usando chaves do mesmo tamanho que as chaves do primeiro. Em prática, isso significa que usando chaves menores, é possível obter a mesma segurança que as grandes chaves do primeiro. Além do maior nível de segurança oferecido, foi mantida a interoperacionalidade com outros comunicadores VoIP, desde que esses utilizem o protocolo SIP para a troca de informações sobre a chamada VoIP e também utilizem o protocolo ZRTP para a troca de chaves simétricas.

A escolha da criação de uma versão 1.2 do componente ZRPT4J se deu em razão da relativa simplicidade na adaptação do SipDroid a versão 1.1 do componente. A escolha do problema de curvas elípticas se deu pela segurança proporcionada pelo mesmo, e pelo

tamanho reduzido das suas chaves.

Ao final deste trabalho, todo o código desenvolvido será proposto como melhorias para os dois projetos *open-source* alterados e utilizados no trabalho: ZRTP4J e SipDroid. Com isso, torna-se possível criar uma maior gama de comunicadores SIP intercomunicáveis com o SipDroid utilizando o algoritmo de Diffie-Helmann com curvas elípticas, além de permitir que outros projetos, não somente comunicadores VoIP, utilizem este algoritmo, e assim possam criar mais conhecimento e desenvolvimento da ciência da computação.

4.1 EXTENSÕES

A seguir são apresentados alguns pontos que podem ser agregados ou melhorados, tanto no SipDroid quanto no ZRTP4J. Como sugestões, seguem:

a) alterar a interface gráfica do SipDroid, permitindo que a verificação do SAS possa ser efetivamente feita e armazenada internamente;
b) criar uma nova API para o ZRTP4J, para que seja simples a sua utilização fora do framework JMF;
c) permitir a utilização de outros algoritmos de geração de números pseudo-aleatórios;
d) alterar o comunicador SIP *Communicator* para que ele possa utilizar a nova versão da biblioteca ZRTP4J;
e) implementar o protocolo ZRTP em um gateway VoIP para comunicação com a rede de telefonia tradicional.

REFERÊNCIAS BIBLIOGRÁFICAS

ACMA. **VoIP quality.** [Canberra], 2009. Disponível em: <http://www.acma.gov.au/WEB/STANDARD/pc=PC_310763>. Acesso em: 07 abr. 2010.

AGÊNCIA NACIONAL DE TELECOMUNICAÇÕES. **Serviços de voz sobre IP (VoIP)**. [Brasília], 2010. Disponível em: <http://www.anatel.gov.br/Portal/exibirPortalPaginaEspecial.do?acao=&codItemCanal=1216&nomeVisao=Cidad%E3o&nomeCanal=Internet&nomeItemCanal=Servi%E7os%20de%20voz%20sobre%20IP%20(VoIP)>. Acesso em: 04 abr. 2010.

ANDROID DEVELOPERS. **What is Android?** [S.l.], 2010. Disponível em: <http://developer.android.com/guide/basics/what-is-android.html>. Acesso em: 04 abr. 2010.

ANDROIDLIB. **Android Market statistics from AndroidLib, Androidlib, Android applications and games.** [S.l.], 2010. Disponível em: <http://www.androlib.com/appstats.aspx>. Acesso em: 05 set. 2010.

ALMEIDA, Juliana. **Transmissão de multimídia multidestinatária.** [Rio de Janeiro], 2003. Disponível em: <http://www.gta.ufrj.br/grad/01_2/vidconf/inicial.html>. Acesso em: 16 maio 2010.

BAZOTTI, Ezequiel. **Voip para computação móvel.** [Cruz Alta], 2007. Disponível em: <http://www.forum.nokia.com/piazza/wiki/images/archive/5/51/20080311210128!TCC_EBazotti_FINAL_3.pdf>. Acesso em: 26 set. 2010.

BIRYUKOV, Alex et al. **Real time cryptoanalysis of a5/1 on a PC.** [New York], 2000. Disponível em: <http://cryptome.org/a51-bsw.htm>. Acesso em: 07 abr. 2010.

BRUNO, Diego S.; DUARTE, Otto C. M. B.; ROTH, Luiz C. B. **Redes de computadores I.** [Rio de Janeiro], 2006. Disponível em: <http://www.gta.ufrj.br/grad/06_1/rtp/>. Acesso em: 16 abr. 2010.

BURNETT, Steve; PAINE, Stephen. **Criptografia e segurança**: o guia oficial RSA. Tradução Edson Furmankiewicz. Rio de Janeiro: Campus, 2002.

COLCHER, Sérgio et al. **VoIP**: voz sobre IP. Rio de Janeiro: Elsevier, 2005.

COURTOIS, Nicolas T.; Piepprzyk, Josef. **Cryptonalysis of block ciphers with overdefined systems of equations.** France. 2002. Disponível em: <http://eprint.iacr.org/2002/044.pdf>. Acesso em: 01 set. 2010.

ELGIN, Ben. Google buys Android for its mobile arsenal. **BusinessWeek,** [New York], 17 Aug. 2005. Disponível em: <http://www.businessweek.com/technology/content/aug2005/tc20050817_0949_tc024.htm>. Acesso em: 05 abr. 2010.

FERGUSON, Niels; SCHROEPPE, Richard; WHITING, Doug. **A simple algebraic representation of Rijndael**. [S.l.], 2001. Disponível em: <http://www.macfergus.com/pub/rdalgeq.pdf>. Acesso em: 20 set. 2010.

FUNDAÇÃO PARA A COMPUTAÇÃO CIENTÍFICA NACIONAL. **H.323.** [Lisboa, Portugal], 2004. Disponível em: <http://www.fccn.pt/index.php?module=pagemaster&PAGE_user_op=view_page&PAGE_id=405&MMN_position=200:5:199>. Acesso em: 04 set. 2010.

GNU TELEPHONY. **GNU ZRTP4J**. [S.l.], 2009. Disponível em: <http://www.gnutelephony.org/index.php/GNU_ZRTP4J>. Acesso em: 29 ago. 2010.

GOOGLE. **The first Android-powered phone**. [Santa Clara], 2008. Disponível em: <http://googleblog.blogspot.com/2008/09/first-android-powered-phone.html>. Acesso em: 05 abr. 2010.

GOOGLE AND BLOG. **Android phones compiled list & faq**. [S.l.], [2009]. Disponível em: <http://www.googleandblog.com/faq-about-google-android>. Acesso em: 05 abr. 2010.

HAMBLEN, Matt. Android to grab no. 2 spot by 2012, says Gartner. **Computerworld,** [S.l.], 6 Oct. 2009. Disponível em: <http://www.computerworld.com/s/article/9139026/Android_to_grab_No._2_spot_by_2012_says_Gartner>. Acesso em: 05 abr. 2010.

HOMMERDING, Roberto; MANSUR, Andre Gonçalves. **Implementação didática de telefone VoIP por software utilizando protocolo SIP**. Curitiba. 2006. Disponível em: <http://www.daeln.ct.utfpr.edu.br/~tcc-daeln/completos2006/implementacaodidatica.pdf>. Acesso em: 04 set. 2010.

INTERNET ENGINEERING TASK FORCE. **Session initiation protocol (SIP)**. [S.l.], 2010. Disponível em: <http://datatracker.ietf.org/wg/sip/charter/>. Acesso em: 04 abr. 2010.

JOHNSTON, Paul. **RSA algorithm**. [Manchester, England], 2010. Disponível em: <http://pajhome.org.uk/crypt/rsa/rsa.html>. Acesso em: 25 set. 2010.

LEOPOLDINO, Graciela M.; MEDEIROS, Rosa C. M. **H.323:** Um padrão para sistemas de comunicação baseado em pacotes**.** [Rio de Janeiro], 2001. Disponível em: <http://www.rnp.br/newsgen/0111/h323.html>. Acesso em: 04 set. 2010.

MALUCHE, André F. **Tecnologias 3g:** cenários e aplicações. 2008. 102 f. Trabalho de Conclusão de Curso (Engenharia de Telecomunicações) – Centro de Ciências Tecnológicas. Universidade Regional de Blumenau, Blumenau. Disponível em: <http://www.bc.furb.br/docs/MO/2008/331678_1_1.pdf>. Acesso em: 07 abr. 2010.

MOREIRA, André. **Criptografia.** Portugal. [2002]. Disponível em: <http://www.dei.isep.ipp.pt/~andre/documentos/criptografia.html>. Acesso em: 05 abr. 2010.

NATIONAL INSTITUTE OF STANDARDS AND TECHOMOLOGY. **Advanced encryption standard**. [S.l.], 2001. Disponível em: <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>. Acesso em: 26 set. 2010.

OLSON, Erik; YU, Woojin. **Encryption for mobile computing**. [Berkeley], [200]. Disponível em: <http://bwrc.eecs.berkeley.edu/Classes/CS252/Projects/Reports/yu_olson.pdf>. Acesso em: 05 abr. 2010.

OLIVEIRA, Júlio M. **MeGaCo:** conheça o protocolo de sinalização de Mídia Gateways VoIP. [São José dos Campos], 2006. Disponível em: <http://www.teleco.com.br/tutoriais/tutorialmegaco/default.asp>. Acesso em: 04 set. 2010.

OPEN HANDSET ALLIANCE. **Industry leaders announce open platform for mobile devices**. [S.l.]. 2007. Disponível em: <http://www.openhandsetalliance.com/press_110507.html>. Acesso em: 05 abr. 2010.

REZENDE, José. **VoIP –** Voice over IP. [Rio de Janeiro], 2004. Disponível em: <http://www.gta.ufrj.br/~rezende/cursos/eel879/trabalhos/voip1/MGCP.html>. Acesso em: 04 set. 2010.

SESSION initiation protocol. In: WIKIPÉDIA, the free encyclopedia. [S.l.]: Wikimedia Foundation, 2010. Disponível em: <http://en.wikipedia.org/wiki/Session_Initiation_Protocol>. Acesso em: 20 maio 2010.

SHEPPARD, Steven. **Curso rápido**: voz sobre IP. Tradução Flávio Morgado. Rio de Janeiro: Ciência Moderna, 2007.

SIPDROID. **FAQ – sipdroid – frequently asked question**. [S.l.], 2009. Disponível em: <http://code.google.com/p/sipdroid/wiki/FAQ#What_SIP_providers_is_Sipdroid_compatible_with?>. Acesso em: 05 set. 2010.

SIPDROID. **Issue 63 – sipdroid – ZRTP support.** [S.l.], 2010. Disponível em: <http://code.google.com/p/sipdroid/issues/detail?id=63>. Acesso em: 06 set. 2010.

SIP-COMMUNICATOR. **SIP communicator**: main – SIP communicator. [S.l.], 2005. Disponível em: <http://www.sip-communicator.org/>. Acesso em: 29 ago. 2010.

WIRESHARK. **Wireshark** – Go deep. [S.l.], 1998. Disponível em: <http://www.wireshark.org>. Acesso em: 01 set. 2010.