UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

CLÁUDIO MANOEL PEREIRA AROUCHA

IMPLEMENTAÇÃO DE MECANISMOS DE SEGURANÇA EM UMA REDE PEER-TO-PEER PARA DISPOSITIVOS MÓVEIS: UM ESTUDO DE CASO DA REDE P2PDROID

São Luís

2012

CLÁUDIO MANOEL PEREIRA AROUCHA

IMPLEMENTAÇÃO DE MECANISMOS DE SEGURANÇA EM UMA REDE PEER-TO-PEER PARA DISPOSITIVOS MÓVEIS: UM ESTUDO DE CASO DA REDE P2PDROID

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Profº. Dr. Mário Antonio Meireles Teixeira

São Luís

2012

Aroucha, Cláudio Manoel Pereira

Implementação de mecanismos de segurança em uma rede peer-to-peer para dispositivos móveis: um estudo de caso da rede P2PDroid/ Cláudio Manoel Pereira Aroucha. – 2012.

60 f.

Impresso por computador (Fotocópia).

Orientador: Mário Antonio Meireles Teixeira.

Monografia (Graduação) – Universidade Federal do Maranhão, Curso de Ciência da Computação, 2012.

1. Redes *peer – to – peer* – Ferramenta – Segurança 2. Dispositivos móveis. 3. Plataforma Android I. Título

CDU 004.056.53

CLÁUDIO MANOEL PEREIRA AROUCHA

IMPLEMENTAÇÃO DE MECANISMOS DE SEGURANÇA EM UMA REDE PEER-TO-PEER PARA DISPOSITIVOS MÓVEIS: UM ESTUDO DE CASO DA REDE P2PDROID

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Profº. Dr. Mário Antonio Meireles Teixeira

Aprovado em: / /

BANCA EXAMINADORA

_____

Prof. Dr. Mário Antonio Meireles Teixeira (Orientador)
Universidade Federal do Maranhão - UFMA

_____

1º Examinador
Prof. Dr. Francisco José da Silva e Silva

Universidade Federal do Maranhão – UFMA

_____

2º Examinador
Profª Msc. Maria Auxiliadora Freire

Universidade Federal do Maranhão - UFMA

Aos meus pais, irmãos, amigos e a toda minha família que, com muito carinho e apoio, não mediram esforços para que eu chegasse até esta etapa de minha vida.

AGRADECIMENTOS

A Deus primeiramente por iluminar e guiar minha vida nesta empreitada, a meus pais Manoel de Jesus e Leila Raquel, as pessoas que me geraram e criaram contra todas adversidades da minha vida conduzindo-me a ser um homem integro que sou. Por sempre enfatizarem e apoiarem meus estudos como algo essencial.

Ao professor e orientador Mário Meireles por ter paciência e sabedoria a orientar-me nesta monografia, prometendo-me um grande trabalho e aprendizado frente a esta tarefa.

Aos meus amigos por serem pacientes a me ensinarem diversas vezes assuntos aos quais não tinha facilidade de aprender, em especial a Letícia Rabêlo, Wallas Henrique, Salviano Lima, Victor Muniz, Renier Pestana e aos outros que aqui não teria espaço para citá-los, além de Thiago Nunes que se empenhou em me por a par de seu trabalho que foi a base desta monografia, meu sincero obrigado a todos.

Maior que a tristeza de não haver vencido é a vergonha de não ter lutado. (RUI BARBOSA)

RESUMO

A utilização e popularidade dos dispositivos móveis nos últimos anos vêm crescendo e as redes P2P como instrumento de compartilhamento de dados também, unir estas duas tecnologias tornou-se viável. Nesse contexto, a plataforma Android sistema operacional de código aberto tornou favorável o desenvolvimento do cliente P2PDroid uma aplicação com esta finalidade, mas este cliente não garante se dados trafegados foram violados. Então para prover que dados compartilhados entre os usuários através deste cliente sejam invioláveis e seguros, utiliza-se uma camada de segurança SSL (Secure Socket Layer).

Palavras-chave: Dispositivos móveis. Segurança. Redes *Peer-to-Peer*. Plataforma Android.

ABSTRACT

The use and popularity of mobile devices have grown in recente years, the P2P networks as a means of sharing data also unite these two technologies became viable. In this context, the Android open source operating system has the favorable customer development P2PDroid an application fo this purpose, but this client does not guarantee that data traffic have been violated. So to provide that data shared between users via this client are inviolable and secure, it uses a security layer SSL (Secure Socket Laier).

Keyword: Mobile devices. Security. Network Peer-to-Peer. Android Plataform.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

ADT	Android Development Tool
API	Application Programming Interface
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
JSSE	Java Secure Socket Extension
MAC	Código de Autenticação de Mensagens
MD5	Message-Digest algorithm5
NIST	National Institute of Standards and Technology
NSA	National Security Agency
OHA	Open Handset Alliance
OSI	Open System Interconnection
SDK	Software Development Kit
SHA	Secure Hash Algorithm
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TTL	Time to live
URI	Uniform Resource Identifier

SUMÁRIO

1 INTRODUÇÃO

Em tempos passados principalmente em guerras utilizavam-se mensagens para comunicar tropas sobre determinadas ordens militares, para que estas não fossem interceptadas pelos inimigos e assim estes interpretarem suas intenções e se prepararem para uma possível "abordagem" elas eram cifradas, codificadas por assim dizer. Dificultar ao máximo a sua interpretação era de suma importância, um algoritmo criptográfico muito antigo chamado cifra de César assim concebido em nome do imperador romano Júlio César fora utilizado para esta finalidade (KUROSE, 2006, p.517). Com este conceito aplica-se à realidade da rede de computadores mundial, a Internet, onde a segurança de dados trafegados é muito valorizada por instituições e corporações financeiras onde estas querem salvaguardar as suas informações. Segundo (GARFINKEL e SPAFFORD,1999) as quatro palavras usadas para descrever todas as diferentes funções criptográficas desempenhadas por sistemas modernos de informação são confidencialidade, autenticação, integridade e não repúdio.

Com o surgimento dos dispositivos móveis e sua grande aceitação e procura pela população, houve a necessidade da implementação de aplicações que satisfizessem as necessidades dos seus usuários. Dado que o sistema operacional Android é uma plataforma baseada em Java com sistema operacional Linux, aberta e livre para estes tipos de dispositivos, torna-se possível realizar esta tarefa (ALVES, 2010). Android é uma plataforma para dispositivos móveis, originada por um grupo de empresas da *Open Handset Alliance* (OHA) liderado pela Google (Android Open Source Project, 2012).

Com o advento do compartilhamento de dados, principalmente arquivos, pela Internet geraram-se as redes P2P, estas que vieram a se consolidar por ser uma solução descentralizada e auto-organizável. A arquitetura cliente-servidor foi a mais utilizada no início da disseminação da Internet por bastante tempo. Este tipo de arquitetura está suscetível a falhas seja sendo por ataques maliciosos ou sobrecarga de requisições tornando o servidor indisponível a uma rede de clientes, além de ter um custo muito elevado. Hoje as redes P2P são responsáveis por mais da metade de fluxo de dados transmitidos pela Internet (BANGEMAN, 2007 apud SILVEIRA, 2009). O cliente P2PDroid é uma aplicação para dispositivos móveis baseado na plataforma Android que proverá todos os recursos que uma rede P2P oferece.

O objetivo é desenvolver uma camada de segurança para o cliente P2PDroid (SOUSA, 2011), a fim de garantir integridade e confidencialidade das informações armazenadas e trafegadas, e ainda prover autenticação de usuários. Para tanto, foi utilizado a tecnologia de *Secure Sockets Layer* (SSL), amplamente disseminada para garantir segurança em aplicações da Internet.

O restante deste trabalho está organizado da seguinte forma, o Capítulo 2 apresenta os principais conceitos e características da plataforma Android, além de expor como é estruturada a sua arquitetura e o ambiente de desenvolvimento de aplicações para a mesma.

O Capítulo 3 apresenta os conceitos da segurança de redes e seus métodos criptográficos. Além de detalhar as camadas de segurança que são utilizadas em instituições comerciais e financeiras.

O Capítulo 4 define a arquitetura do cliente P2PDroid, operações definidas, sua aplicação e a estrutura dos arquivos de metadados utilizados.

O Capítulo 5 apresenta a implementação da camada de segurança para o cliente P2PDroid, além de demonstrar a arquitetura, as API's e ferramentas que foram utilizados para seu desenvolvimento.

O Capítulo 6 aponta os resultados deste trabalho, as limitações encontradas e possíveis trabalhos futuros que agregariam mais valor ao mesmo.

2 ANDROID

A plataforma Android é um software desenvolvido pela Google (Google, 2012). É mantida por um grupo de mais de 40 empresas na OHA (OHA, 2012) que criaram como resposta às próprias experiências com lançamentos de aplicativos, onde tinham como objetivo uma plataforma aberta, disponível para operadoras, fabricantes e desenvolvedores para desta forma inovarem neste campo de dispositivos móveis, beneficiando assim os usuários finais (Android Open Source Project, 2012). Neste capítulo apresentaremos as características, arquitetura e o desenvolvimento de aplicações para esta plataforma.

2.1 Características da Plataforma Android

A plataforma Android é baseada em Java com sistema operacional Linux, kernel 2.6, projetada para dispositivos móveis, além de ser um rico ambiente para desenvolvedores, com recursos (sistema de localização, rede sem fio, *Bluetooth*, etc) somente sendo limitado pelo hardware.

Os desenvolvedores foram atraídos a produzirem e publicarem aplicações baseados no Android, por haver uma loja *online* chamada Google *Play*(Google *Play*, 2012) que os vendia e distribuía gratuitamente, desta forma tornando os dispositivos móveis desta plataforma muito procurados pela sua diversidade de aplicativos.

2.2 Arquitetura do Android

A arquitetura do ambiente de desenvolvimento Android é dividida em 5 camadas, conforme figura 2.1.

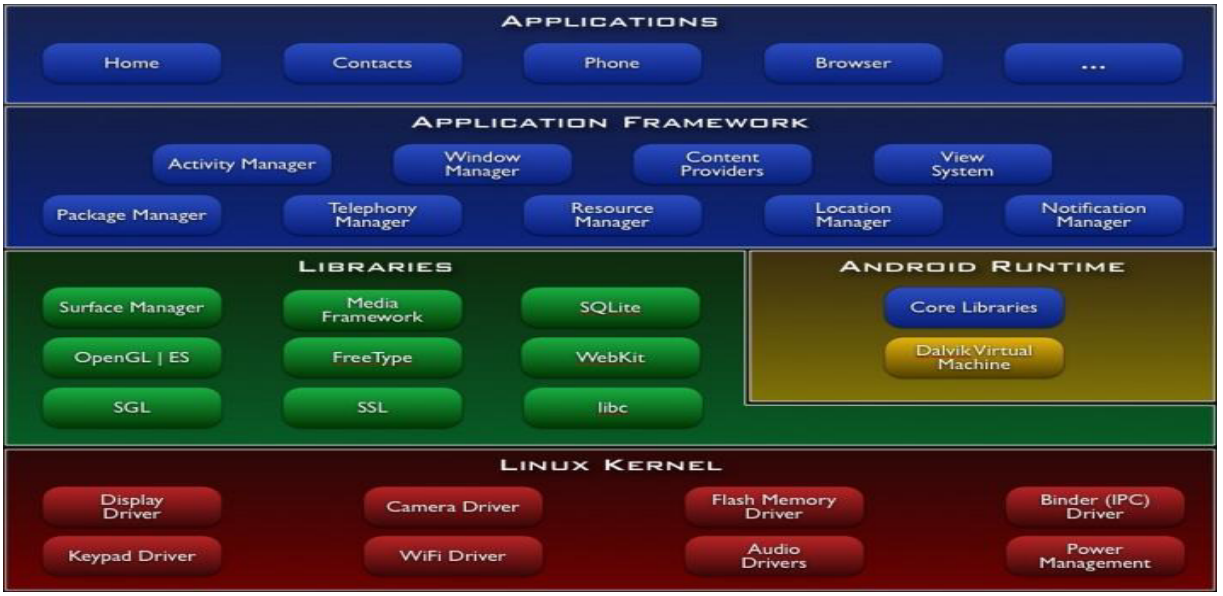

Figura 2.1 Arquitetura Android (Android Developers, 2012)

Na camada superior estão as aplicações que são escritas em linguagem Java, por exemplo: cliente de email, calendário, contatos, etc. Abaixo desta camada, vêm os componentes de *framework*, formando uma API (*Application Programming Interface*) para ser utilizada por outras aplicações como definir alarmes, adicionar notificações na barra de *status* do dispositivo, etc (Android Developers, 2012).

No terceiro nível da camada encontram-se as bibliotecas, escritas em C/C++, usadas por diversos componentes do sistema Android. Exemplo de algumas das bibliotecas são SQLite, SSL, *Surface Manager*, WebKit, SGL, libc e outros.

Na quarta camada de sua arquitetura funciona o ambiente de tempo de execução (Android *Runtime*) que inclui um conjunto de bibliotecas que fornece a maioria das funcionalidades disponíveis na linguagem de programação Java, além de uma máquina virtual para executar aplicações, chamada VM Dalvik (DalvikVM, 2012). A VM Dalvik executa os arquivos em formato executável Dalvik (.dex) otimizado para a pouca memória que um dispositivo móvel geralmente dispõe e estes arquivos .dex são compactados em pacotes de extensão apk, a forma como as aplicações Android são distribuídas.

A quinta camada é a base da arquitetura, uma camada de abstração que se situa entre o hardware e o restante da pilha de software, utilizando o Linux kernel versão 2.6 para serviços centrais, como segurança, gerenciamento de memória, gestão de processos, pilha de rede e modelo de driver (ALVES, 2012).

2.3 Desenvolvimento de Aplicações

Tudo de que se necessita para criar aplicações é do Android SDK (*Software Development Kit*) e a API compatível com a versão do sistema operacional instalado no dispositivo móvel. Todos os aplicativos são escritos em linguagem Java e os arquivos e códigos são compactados em um único pacote de formato .apk, depois de compilados, arquivo que o Android utiliza para instalar a aplicação.

2.3.1 Android SDK

A plataforma Android fornece o SDK, um *kit* para desenvolvimento de aplicações que inclui documentação, código e utilitários, além de bibliotecas, *frameworks*, ferramentas para *debug* e emulador de dispositivo móvel.

A IDE (*Integrated Development Environment)* recomendada para desenvolvedores de aplicativos Android é o Eclipse (Eclipse, 2012), com o *plugin* ADT (*Android Development Tool*) amplia os seus recursos auxiliando a criar rapidamente novos projetos Android, uma interface de aplicativo, adicionar os pacotes baseados no Android *Framework* API, depurar seus aplicativos usando as ferramentas do Android SDK e até mesmo exportar de forma assinada (ou não assinada) os arquivos .apk a fim de distribuir sua aplicação e editores de *Extensible Markup Language* (XML, 2012) personalizados , dentre outros recursos.

O Android SDK é composto por diferentes níveis de API referentes a cada versão da plataforma. Segundo (Android Developers, 2012), as API's de versões anteriores geralmente são compatíveis com as mais modernas, mas o contrário não. Se for necessário o desenvolvimento de uma aplicação para uma versão antiga do Android deve-se fazer o *download* do nível da API a partir do SDK para então desenvolver a aplicação.

2.3.2 Componentes dos Aplicativos

Componentes dos aplicativos são blocos de construção essenciais para o

desenvolvimento de uma aplicação Android. Há quatro tipos de componentes distintos: *Activity, Service, Broadcast Receiver* e *Content Provider*. Cada um com uma finalidade e ciclo de vida distinto definindo como o componente é criado e destruído.

A *Activity*, ou atividade, representa uma tela onde usuário pode realizar algo. Uma *Activity* é da ordem de uma a várias telas no mesmo programa. É necessário definir a primeira tela a ser exibida ao usuário chamada de atividade "principal". Toda *activity* pode iniciar outra que realize operações diferentes, sendo que a navegação é guiada por *Intents*, que especificam tanto a atividade exata que se deseja iniciar ou descrever o tipo de ação que realiza, além de transportar pequenas quantidades de dados que você deseja utilizar na próxima atividade a ser iniciada. Estas atividades seguem um ciclo de vida que será apresentado na seção 2.3.3.

Services é um componente que executa em segundo plano processos remotos ou operação de longa duração. Este serviço não oferece interface de usuário.

O *Broadcast Receiver* é o componente que responde a todos os anúncios do sistema, por exemplo, uma transmissão que anuncia que a bateria está fraca, ou que uma imagem foi capturada, gera um alerta ao usuário que ocorreu um evento de transmissão. Assim como nas atividades, os objetos *Services* e o *Broadcast Receiver* são iniciados por *Intents*.

Content Provider é um provedor de conteúdo que gerencia um conjunto compartilhado de dados. Através dele as aplicações podem consultar ou até mesmo alterar dados, sejam eles armazenados em um sistema de arquivos, banco de dados SQLite, Web, ou qualquer outro local de armazenamento persistente. Através de uma URI (*Uniform Resource Identifier*) pública identifica-se os dados em um provedor com que queira trabalhar. O Android define constantes CONTENT_URI para todos os provedores que vêm com sua plataforma. A constante URI é usada em todas as interações com o *content provider*. Por exemplo, para acessar os contatos do usuário armazenado em uma aplicação, utilizamos a URI "*content://contacts/people*". O "*content://*" é a sequência que sempre está presente e identifica a URI, o *contacts* ou *user_dictionary* é a autoridade do provedor e o *people* é o caminho da tabela.

A seguir, a figura 2.2 exemplifica o acesso a dados fornecidos por terceiros.

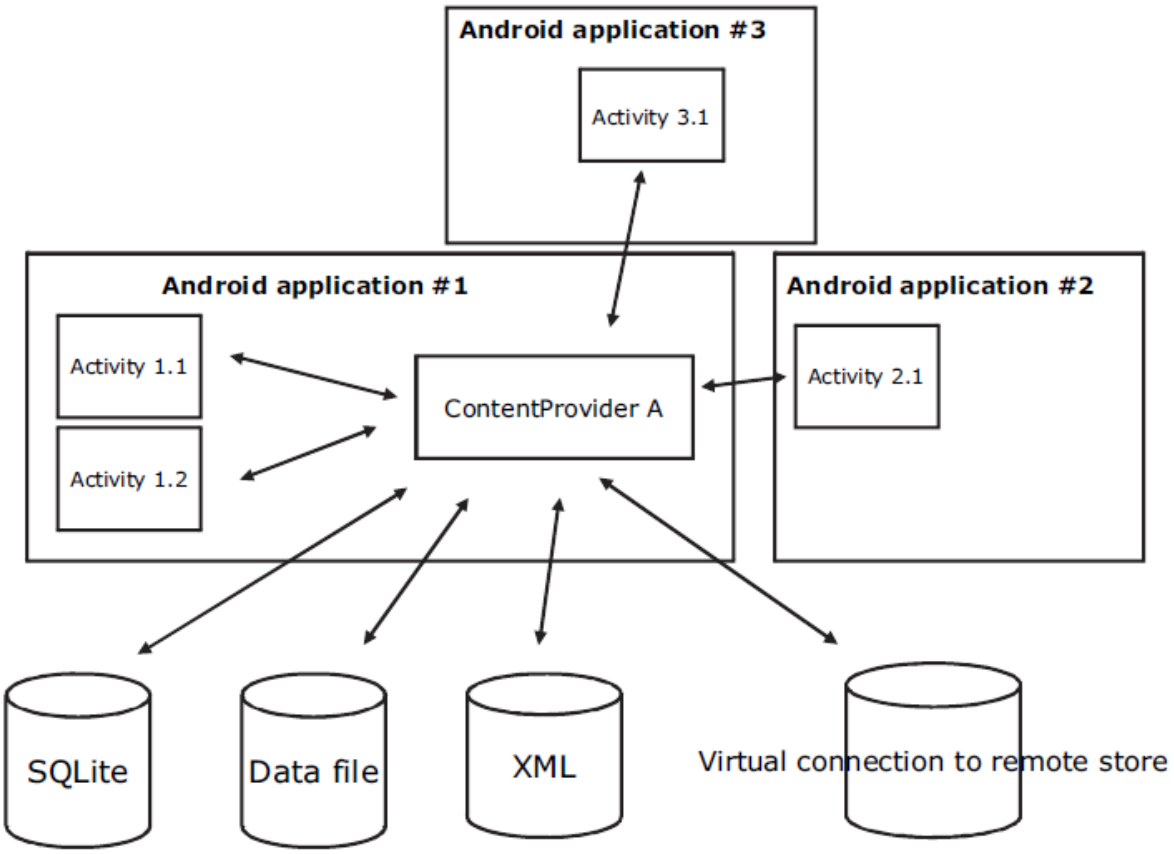

Figura 2.2 Acesso a dados via *ContentProvider* (ABLESON et al, 2009)

2.3.3 Ciclo de Vida das Atividades

Há três tipos básicos de estado de uma atividade: *resumed* (retomar), *paused* (pausado) e *stopped* (parado). Para poder gerenciar as atividades e seus estados utiliza-se a *Activity Stack*, uma pilha que preserva as atividades.

No estado *resumed,* caso a atividade esteja executando ela está em primeiro plano da tela e tem o foco do usuário. Caso *paused*, a atividade é parcialmente transparente, mas continua ativa. No estado *stopped*, a atividade terá sido obscurecida totalmente ou sobreposta por outra.

A transição entre os estados citados acima é dada pela chamada aos métodos *onCreate()*, *onRestart()*, *onStart()*, *onResume()*, *onPause()*, *onStop()* e *onDestroy()* como pode ser observado na Figura 2.3. Se uma atividade está em *paused* ou *stopped*, o sistema pode removê-la da memória, quer seja pedindo para terminar (chamando seu método *finish()*), ou simplesmente encerrando o seu processo caso necessite de memória. Quando a

atividade é aberta novamente (depois de ter sido finalizada ou encerrada), ela deve ser criada e colocada na pilha de execução.

A seguir a figura 3.3 que exemplifica o funcionamento do ciclo de vida de uma *Activity*.

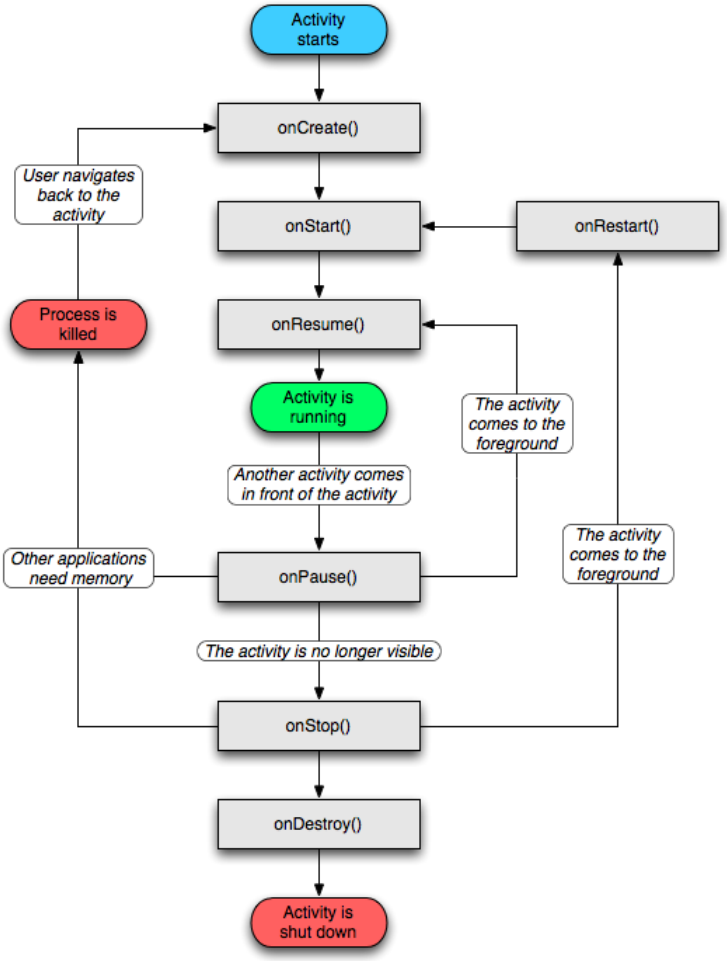

Figura 2.3 Ciclo de vida de uma atividade (Android Developers, 2012)

2.3.4 O arquivo AndroidManifest.xml

Cada aplicação deve ter um arquivo AndroidManifest.xml em seu diretório raiz, pois é neste arquivo que se encontram as principais informações da aplicação. Descrevem os componentes (*Activity*, *Service*, *BroadcastReceiver* e *ContentProvider*), além dos *Intent*s e *IntentFilters*, declara as permissões que uma aplicação dispõe para interagir com partes

protegidas de uma API ou aplicações, declara o nível mínimo da API do Android, determina quais os processos irão sediar os componentes da aplicação e guarda o nome da aplicação, tudo em formato XML (XML, 2012).

As aplicações são executadas de forma separada e independente, em um modelo chamado de *sandbox*, ou caixa de areia, onde é permitido utilizar tudo que esteja dentro da caixa, mas é necessário autorização para utilizar o que estiver do lado de fora. Cada aplicação instalada é associada a um *user* ID Linux único o qual será utilizado na criação da sua *sandbox*. Visto isto duas aplicações distintas não podem ser executadas no mesmo processo, pois não possuem os mesmos ID's, exceto quando os identificadores são compartilhados, sendo tratados como a mesma aplicação. Para obter ID's compartilhados deve-se declarar no arquivo manifesto um *sharedUserID* (Sachse, 2010 apud Sousa, 2011).

A Figura 2.4 exemplifica um arquivo AndroidManifest que possui uma atividade principal chamada de "*Main*" e que permite acesso externo pela aplicação, com a declaração da permissão "*android.permission.INTERNET*".

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="p2pws.p2pwsdroid" android:versionCode="1" android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".Main" android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-permission android:name="android.permission.INTERNET"></uses-permission>
</manifest>
```

Figura 2.4 Exemplo do AndroidManifest.xml (SOUSA, 2011, p. 23)

2.3.5 Interface com o usuário

As interfaces do usuário são construídas usando objetos *View* e *ViewGroup*. Estes objetos são definidos hierarquicamente em forma de uma árvore na plataforma Android, como pode ser visualizado na figura 2.5.

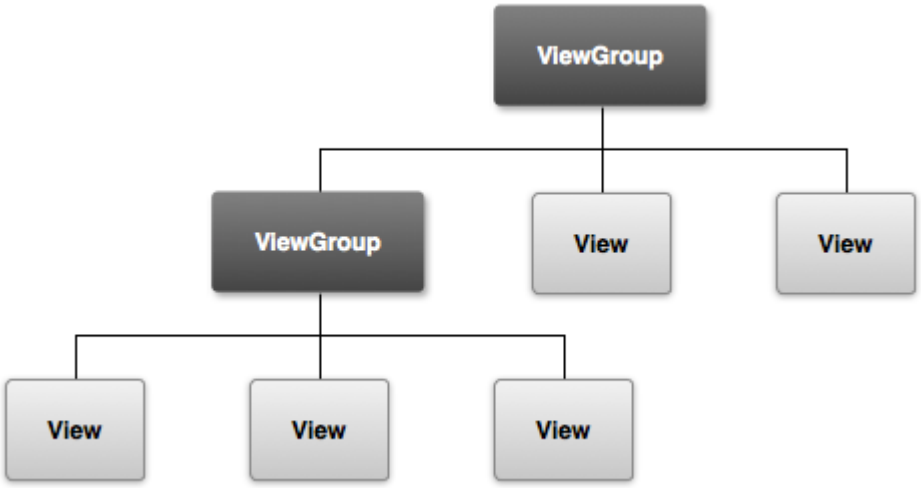

Figura 2.5 Hierarquia de uma View e ViewGroup (Android Developers, 2012)

Objetos *View* são as folhas da árvore e servem como base para as subclasses chamadas de *widgets*, que oferecem objetos de interface do usuário já implementados, por exemplo, como botões, barras de *status*, etc. Já os *ViewGroups*, são os ramos da árvore, a base para as subclasses chamadas "*layouts*", sendo estes diversos em sua arquitetura como linear, tabular ou relativo.

A forma mais comum para definir o *layout* e expressar a hierarquia é utilizando um arquivo XML de *layout*, além de definir sua hierarquia de *Views* e *ViewGroups*. Na figura 2.6 demonstra um arquivo XML de *layout*, definindo um *layout* linear e três *widgets*, um *TextView*, um *EditText* e um *Button*. A Figura 2.7 mostra como esta interface é exibida na tela do emulador.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:id="@+id/widget30"
    android:layout_width="fill_parent" android:layout_height="fill_parent" android:padding="10px"
    android:orientation="vertical" xmlns:android="http://schemas.android.com/apk/res/android">

    <TextView android:id="@+id/txtName" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="@string/txtName">
    </TextView>
    <EditText android:id="@+id/edTxtNome" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:editable="false" android:focusable="false">
    </EditText>
    <Button android:id="@+id/btnSave" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="@string/btnSave">
    </Button>
</LinearLayout>
```

Figura 2.6 Exemplo de arquivo XML de *layout* (SOUSA, 2011, p. 24)

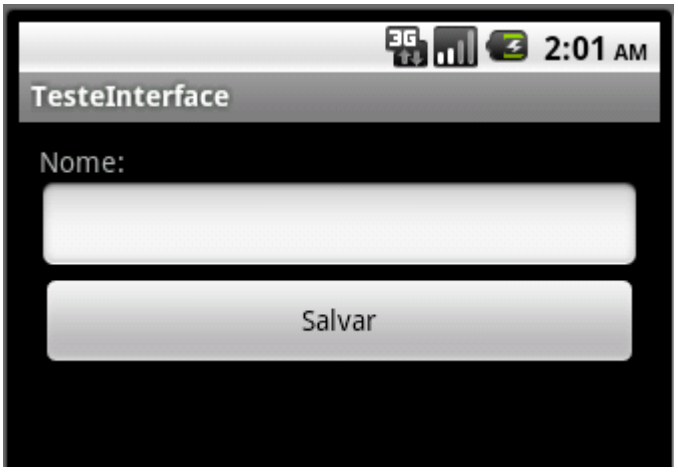

Figura 2.7 Exemplo de interface desenvolvida utilizando XML (SOUSA, 2011, pag. 24)

2.4 Considerações Finais

Segundo (GARTNER, 2012), os dispositivos baseados na plataforma Android irão comandar 49 por cento de todo o mercado de venda de *smartphones* até o final de 2012, dada a grande diversidade de dispositivos móveis para esta plataforma, além do seu custo ser menor e ter um amplo suporte oferecido pela Google.

Com um ambiente rico em ferramentas, torna-se atrativo para os desenvolvedores escreverem seus aplicativos, comercializando-os ou distribuindo-os gratuitamente em uma loja *online* como a Google *Play* (Google Play, 2012), motivando desta forma os usuários finais na aquisição de um dispositivo móvel baseado em Android pela vasta quantidade de aplicativos ali disponível.

3 SEGURANÇA DE REDE

Nos dias de hoje a Internet tem sido um dos grandes veículos de comunicação mundial tanto que no âmbito nacional vem crescendo de forma vertiginosamente seja pela necessidade ou pelo incentivo do governo federal, pois segundo (CGI.br, 2010) planos de incluir a Internet para mais de 88 por cento das residências do Brasil estão sendo discutidos.

Os dados transmitidos por esta rede de computadores tornam-se vulneráveis a ponto de serem interceptados por usuários indevidos e não autorizados, sendo necessário um modo de garantir que essas informações não sejam modificadas, observadas ou recebidas por terceiros, utiliza-se a criptografia como mecanismo de segurança essencial para garantir a integridade, confidencialidade e autenticidade das mesmas.

3.1 Criptografia Simétrica

Segundo (STALLINGS, 2008), até a década de 1970 a criptografia simétrica era o único tipo de criptografia conhecida, onde se utilizava uma única chave secreta e um algoritmo de criptografia para transformar um texto claro em um texto cifrado, e que a partir do algoritmo de descriptografia e a mesma chave de criptografia conseguiria recuperar do texto cifrado o texto claro.

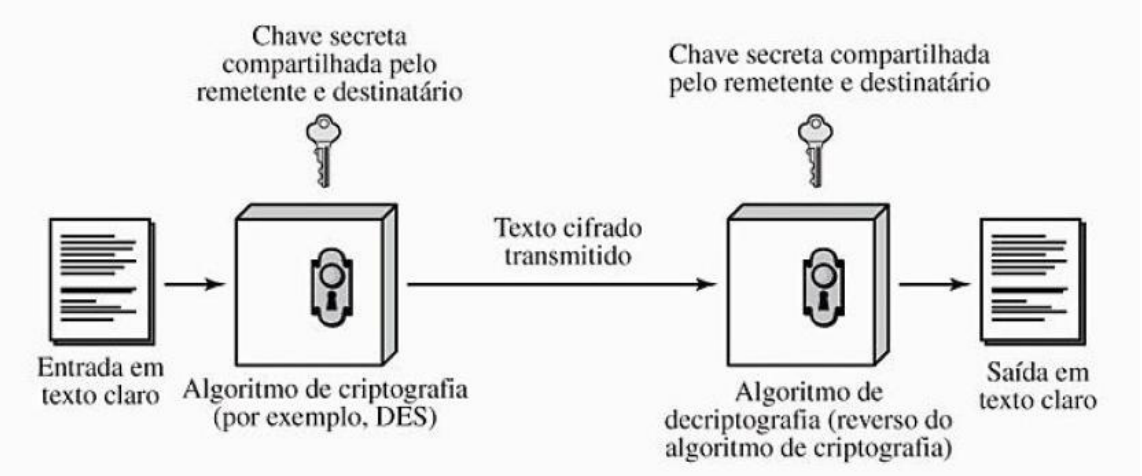

Figura 3.1 Modelo simplificado de Criptografia Simétrica (STALLINGS, 2008, p.18).

Os algoritmos de criptografia ou descriptografia não precisam ser secreto o que torna a criptografia simétrica praticável, a dificuldade está em manter a chave secreta, pois ela é a única informação sigilosa que impede um intruso de descriptografar os dados transmitidos (KUROSE, 2006). Um dos maiores questionamentos é como realizar a troca da chave secreta entre o emissor e receptor de forma segura e garantir a proteção da mesma contra terceiros (SOUSA, 2009).

Um exemplo de esquema de comunicação simétrica é compartilhar uma chave simétrica K por um canal seguro tanto para o emissor e receptor de uma mensagem. Do lado da fonte da mensagem X, criptografamos com o algoritmo E com a chave K produzindo desta forma um texto cifrado Y.

$$Y = E(X, K)$$

Do lado do receptor realiza-se a descriptografia com o algoritmo D no texto criptografado com a chave secreta K anteriormente compartilhada obtendo assim a mensagem X.

$$X = D(Y, K)$$

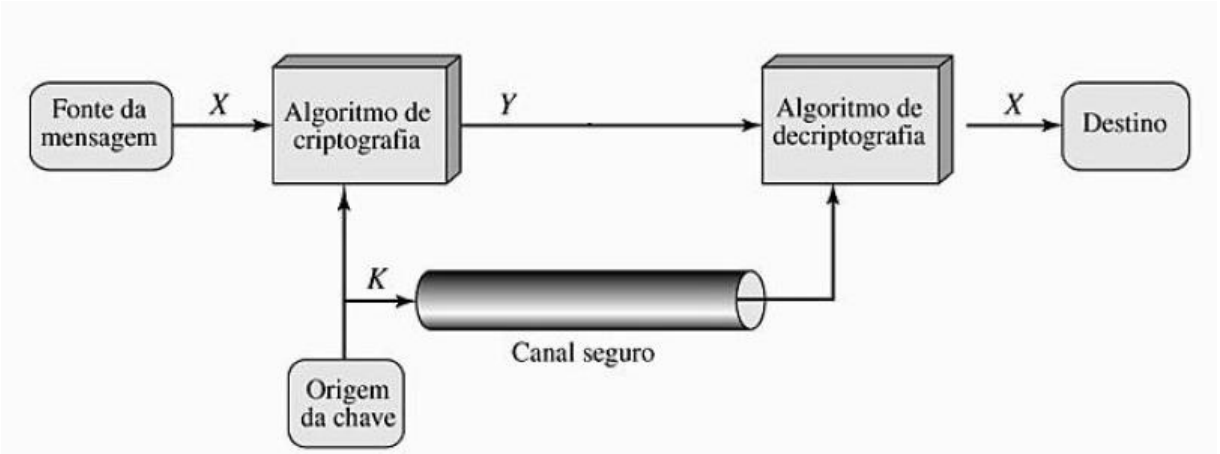

Figura 3.2 Modelo de Criptografia Simétrica. Adaptada (STALLINGS, 2008, p.19).

Um dos principais algoritmos de criptografia simétrica é o DES (*Data Encryption Standard*) desenvolvido em 1977, que utiliza uma string alfanumérica para criptografar e descriptografar dados transmitidos em uma rede de computadores. O DES utiliza mais de um cifrário para criptografar uma mensagem, o que assegura uma maior segurança (SOUSA, 2009). Segundo o *National Institute of Standards and Technology* (NIST, 1999 apud KUROSE, 2006), "O objetivo é embaralhar completamente os dados e a chave, de modo que todos os bits do texto cifrado dependam de todos os bits de dados e de todos os bits da chave

(...) com um bom algoritmo, não deverá haver nenhuma correlação entre o texto cifrado e os dados originais e a chave".

Uma chave DES é composta de 64 dígitos binários ("0" s ou "1" s), dos quais 56 bits são gerados aleatoriamente e usado diretamente pelo algoritmo e os outros 8 bits são usados para detectar erros.

3.2 Criptografia Assimétrica

O algoritmo de criptografia simétrica requer que a distribuição da chave seja feita por um canal seguro, a dificuldade está em assegurar o compartilhamento com sucesso. A solução proposta é utilizar a criptografia assimétrica, pois ela consiste em utilizar um par de chaves, uma para criptografar (chave pública) e outra para descriptografar (chave privada). A criptografia assimétrica é também conhecida como criptografia de chave pública (SOUSA, 2009).

Em um exemplo de utilização de chaves públicas (KUROSE, 2006), Alice e Bob querem se comunicar pelos computadores através da Internet, Bob possui uma chave pública (K_b^+) que é de conhecimento de todos e uma privada (K_b^-) que somente ele a conhece.

Para se comunicar com Bob, Alice primeiramente busca a chave pública dele e utiliza um algoritmo de criptografia, padronizado e conhecido, para desta forma codificar uma mensagem *m*, obtendo $K_b^+(m)$. Após a mensagem ser enviada a Bob, este utiliza um algoritmo de descriptografia (padronizado) e sua chave privada (K_b^-), $K_b^-\ (K_b^+\ (m))$, para assim poder obter a mensagem *m* de Alice.

Este método permite que Alice e Bob se comuniquem sem que haja troca de uma chave secreta antecipadamente. Conforme a figura 3.3.

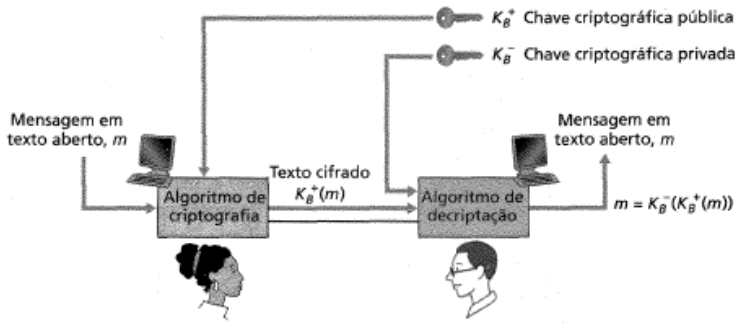

Figura 3.3 Modelo de Criptografia de chave pública (KUROSE, 2006, p.522).

Para obter o par de chaves gera-se primeiramente um número aleatório o qual será a chave privada e a partir deste será calculado a chave pública, o processo de criação das chaves é unidirecional, ou seja, é matematicamente complexa e computacionalmente inviável, a obtenção da chave privada a partir da pública.

Um dos algoritmos de chave pública mais popular é o RSA cujo nome é baseado nos seus inventores Rivest, Shamir e Adleman. Segundo (KUROSE, 2006), este algoritmo funciona da seguinte forma: escolhem-se dois números primos grandes *p* e *q*, quanto maiores mais seguros serão as mensagens a serem codificadas, obtem-se o número composto *n* a partir da multiplicação de *p* e *q*, $n = p * q$, calcula-se *z*, $z = (p - 1)*(q - 1)$, deve ser escolhido um número *e* menor que *n* e que não possua fator comum com *z* além de 1, para utilizá-lo na criptografia. Para a descriptografar é usado o número *d* tal que $e*d - 1$ seja divisível por *z*, a chave pública, K_b^+, sendo este o par de números (*n, e*) e a privada, K_b^-, par de números (*n*, *d*).

Um exemplo prático, Alice envia um número *m*, para codificar a mensagem utiliza-se a chave pública de Bob (K_b^+) o par de números (*n*, *e*), obtendo assim um texto cifrado *c*.

$$c = m^e \ mod \ n.$$

Para decifrá-lo Bob utiliza sua chave privada (K_b^-), par de números (*n*, *d*), no texto cifrado *c*, obtendo *m* no processo.

$$m = c^d \ mod \ n.$$

Uma etapa de criptografia de mensagem seguida de uma descriptografia obtém-se m^{e*d}. Utilizando a teoria dos números se *p* e *q* forem primos e $n = p*q$, então $x^y \ mod \ n$ é igual a $x^{y \ mod \ (p-1)*(q-1)} \ mod \ n$, o qual $y \ mod \ (p-1)(q-1)$ *é igual a 1*.

$$m^{e*d} \ mod \ n = \ m^{(e*d \ mod \ (p-1)*(q-1))} \ mod \ n = m.$$

Como *e*d* é divisível por (p-1)*(q-1) com resto 1, então $e*d \ mod \ (p-1)*(q-1) = 1$.

$$m^{e*d} \ mod \ n = \ m^1 \ mod \ n = m,$$

ou seja,

$$m^{e*d} \ mod \ n = \ m.$$

A ordem da sequência da potência *e* e *d* não interfere no resultado, isto é se inverter a ordem da criptografia e descriptografia, obtém-se de qualquer forma o valor original *m*.

$$m^{e*d} \ mod\ n = \ m = \ m^{d*e} \ mod\ n$$

A segurança do RSA reside no não conhecimento de um algoritmo rápido que a partir do valor *n* público possa decompô-lo em números primos *e* e *d.*

3.2.1 Assinatura Digital

A assinatura digital é a forma de provar se determinada mensagem é de determinada pessoa e não de um terceiro passando-se por ela. Utiliza-se o processo de criptografar uma mensagem *m* com a chave privada K_b^- do remetente, $K_b^-\ (m)$, o receptor da mensagem somente poderá descriptografá-la com a chave pública do remetente, caso seja realizado comprova-se a autenticidade do remetente e a integridade do seu conteúdo. Este é um método que é verificável, não falsificável e não repudiável, pois somente uma pessoa poderia conhecer a chave privada K_b^- e gerar a chave pública, (K_b^+), admitindo que *b* não repassou a sua chave ou não fora roubado (KUROSE, 2006).

$$K_b^+\ (K_b^-\ (m)) = m$$

Porém a mensagem, $K_b^-\ (m)$, não remete à confidencialidade pelo fato de qualquer pessoa conhecendo o algoritmo de descriptografia e a chave pública do remetente verificar o seu conteúdo, a solução é aplicar do lado do emissor a criptografia de chave pública do destinatário (K_a^+), desta forma assegura que somente o receptor que detém a chave privada (K_a^-) poderá descriptografá-la e obter a mensagem *m*.

$$K_a^-\ (K_a^+\ (K_b^-\ (H(m)))) = m$$

3.2.2 Funções de Hash

Criptografar e descriptografar mensagens são processos muito dispendiosos, utilizar um resumo da mensagem original para ser criptografado é mais viável. Existem vários algoritmos para calcular o resumo de uma mensagem, conhecidos como funções de *Hash* (*H*), mas devem ter as seguintes propriedades: duas mensagens diferentes (*x* e *y*) devem ser inviáveis computacionalmente de produzirem, *H(x) = H(y)*, além de ser simples o cálculo (KUROSE, 2006).

Para poder proteger a mensagem original, seu resumo de mensagem é assinado digitalmente não podendo ser alterado por ninguém, $K_b^-(H(m))$, para verificar se a mensagem *m* recebida pelo destinatário não fora modificada, $(m, K_b^-(H(m)))$, por um intruso ou por acidente é utilizado a função de *Hash* na mensagem *m* e comparada com o conteúdo da mensagem descriptografada com a chave pública do emissor, caso sejam iguais a mensagem enviada é íntegra e não repudiável.

Alguns algoritmos de *Hash*: o MD5(*Message-Digest algorithm5*) desenvolvido pelo Rivest em 1991 (RSA, 2012) e o SHA-1 (SHA-1, 2012) projetado pela National Security Agency (NSA, 2008) e pelo NIST.

3.3 Certificado Digital

Certificado digital é um conjunto de informações de uma entidade (empresa, pessoa física ou até um computador) juntamente com a chave pública associada, além da chave privada que somente a entidade especificada no certificado a detém.

O problema do algoritmo de chaves públicas pode ser exemplificado a seguir, Alice está comunicando-se com Bob, como ter certeza que a chave pública de Bob que detém seja realmente dele. Para dar confiabilidade a esta chave pública utilizamos uma autoridade certificadora (*certification authority* - CA), cuja função é validar e emitir certificados. Admitindo que a chave pública da CA, K_{CA}^+, seja de conhecimento de todos, por exemplo, esteja disponibilizada em local público de confiança, e não há como ser falsificada. Alice utiliza a *CA* para poder confirmar que a suposta chave pública de Bob que detém realmente seja dele. Como pode ser visto na figura 3.4 a chave pública de Bob está certificada pela *CA*

agora esta pode ser distribuída em qualquer lugar, seja sendo em uma página Web ou servidor de chaves públicas (KUROSE, 2006).

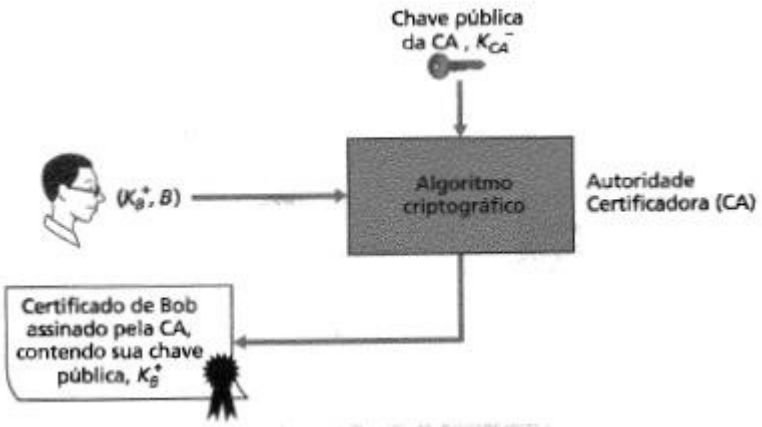

Figura 3.4 Certificado de Bob obtido de uma CA (KUROSE, 2006, p.540).

> A escolha de confiar em uma CA é similar ao que ocorre em transações convencionais, que não se utilizam do meio eletrônico. Por exemplo, uma empresa que vende parcelado aceita determinados documentos para identificar o comprador antes de efetuar a transação. Estes documentos normalmente são emitidos pela Secretaria de Segurança de Pública e pela Secretaria da Receita Federal, como o RG e o CPF. Existe, aí, uma relação de confiança já estabelecida com esses órgãos. Da mesma forma, os usuários podem escolher uma CA à qual desejam confiar a emissão de seus certificados digitais. (ITI, 2012, p. 11).

3.4 Sockets Seguros (Secure Sockets Layer - SSL)

A Secure Socket Layer (SSL) foi desenvolvida pela Netscape em 1994, mas somente lançada em 1996 na sua versão 3.0 (MOZILLA, 2012). Este protocolo foi projetado para utilizar TCP (*Transmission Control Protocol*) para fornecer um serviço de segurança confiável fim a fim (STALLINGS, 2008). A camada SSL situa-se entre a camada de aplicação e a de transporte (KUROSE, 2006) do *modelo Open System Interconnection* (OSI). A SSL é independente do protocolo utilizado seja HTTP, FTP, etc. Esta não é limitada à Web através de *browsers*, podendo ser implementada em servidores e aplicativos para comércio na Internet.

Os mecanismos ofertados pela SSL são:

- Autenticação: as partes comunicantes, servidor e cliente, podem se autenticar mutuamente ou opcionalmente através de algoritmos de chaves públicas (RSA).
- Confidencialidade: é criada uma sessão SSL, que criptografa todos os dados transferidos

pelo remetente por meio de chaves secretas e algoritmos simétricos (DES), estes definidos durante a troca de chaves na primeira fase do estabelecimento da sessão.

- Integridade: para assegurar a integridade das mensagens transmitidas é utilizado códigos de autenticação de mensagens (MAC) calculados através de algoritmos de funções de *Hash* (MD5 ou SHA) (CABRAL, 2002 apud SOUZA, 2008).

Segundo (STALLINGS, 2008), SSL tem dois conceitos fundamentais:

- Sessão SSL: é a associação entre o cliente e servidor, criado pelo protocolo de estabelecimento de conexão. Serve para definir os parâmetros criptográficos compartilhados entre várias conexões.
- Conexão SSL: é um transporte apropriado a um tipo de serviço. As conexões são ponto a ponto e estão associadas a uma sessão.

3.4.1 Arquitetura e Estabelecimento da Comunicação

A arquitetura SSL é composta por duas camadas de protocolos diferentes, a primeira tem-se o SSL *Record Protocol* e a segunda o *Change Cipher*, *Alert Protocol* e *Handshake Protocol*. A figura 3.5 demonstra os protocolos SSL.

Camadas TCP/IP com SSL	
	Camada de Aplicação HTTP, FTP, TELNET...
SSL	Change cipher, Alert Protocol, Handshake Protocol
SSL	Camada SSL Record Protocol
	Camada TCP
	Camada IP

Figura 3.5 Pilha de protocolos do SSL

SSL *Record Protocol*: fragmenta cada mensagem da camada superior em blocos de 2^{14} $bytes$ ou menos transformando-os em registros *SSLPlaintext*, realiza compressão de *SSLPlaintext* para registros *SSLCompressed* sem que haja perda de dados e aumento de no máximo 1024 *bytes,* o algoritmo utilizado para compressão foi negociado durante o *handshake*, o valor padrão é nulo para o SSL na versão 3.0. A etapa seguinte utiliza a chave secreta compartilhada durante o *handshake* para calcular o MAC (Código de Autenticação de

Mensagens) sobre os dados compactados. A autenticação da mensagem compactada é acrescida do MAC para posteriormente realizar a criptografia de chaves simétricas (as funções criptográficas foram definidas durante o *handshake*), transformando *SSLCompressed* em *SSLCiphertext*. A última etapa é definir o cabeçalho: o tipo de conteúdo (protocolo da camada mais alta que processa o fragmento transportado), versão principal (versão utilizada caso seja SSLv3, valor 3), versão secundária (caso SSLv3, valor 0) e o tamanho do conteúdo (tamanho em *bytes* do fragmento do texto claro).

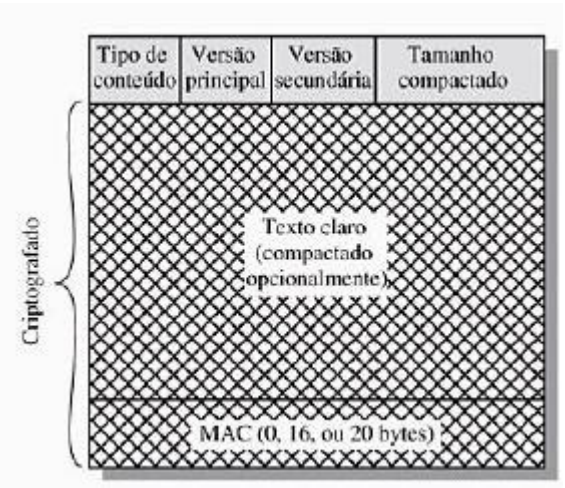

Figura 3.6 Formato do registro SSL (STALLINGS, 2008, p. 384).

SSL *Change Cipher*: é composto por uma única mensagem de um 1 *byte* sua funcionalidade é atualizar o conjunto de cifras a ser utilizado na conexão.

SSL *Alert Protocol*: utilizado para enviar alertas sobre o SSL para as partes envolvidas. As mensagens de alerta são compactadas e criptografadas, é composta por dois *bytes*. O primeiro *byte* identifica o grau de gravidade da mensagem podendo ser do tipo *warning* ou fatal, caso seja fatal o SSL encerra a conexão. O segundo *byte* contém o código de um tipo de alerta específico (erros de certificados, de criptografia ou sequências de mensagem).

SSL *Handshake Protocol*: realiza a autenticação entre cliente e servidor, negocia o Código de Autenticação de Mensagens (MAC), algoritmos criptográficos e chaves que utilizarão para proteger os dados transmitidos. Utiliza a criptografia de chaves públicas para realizar a negociação inicial, onde a chave simétrica de sessão gerada aleatoriamente é transferida por um meio seguro. Para garantir uma maior segurança é utilizado a função *hash* (MD5 ou SHA) conjuntamente com o MAC. O protocolo *Handshake* é utilizado antes de qualquer transferência de dados seja realizada.

Uma das formas mais genéricas para o estabelecimento da comunicação é descrito em capacidades de segurança, autenticação do servidor, autenticação do cliente e término. As 4 fases são descritas na figura 3.7.

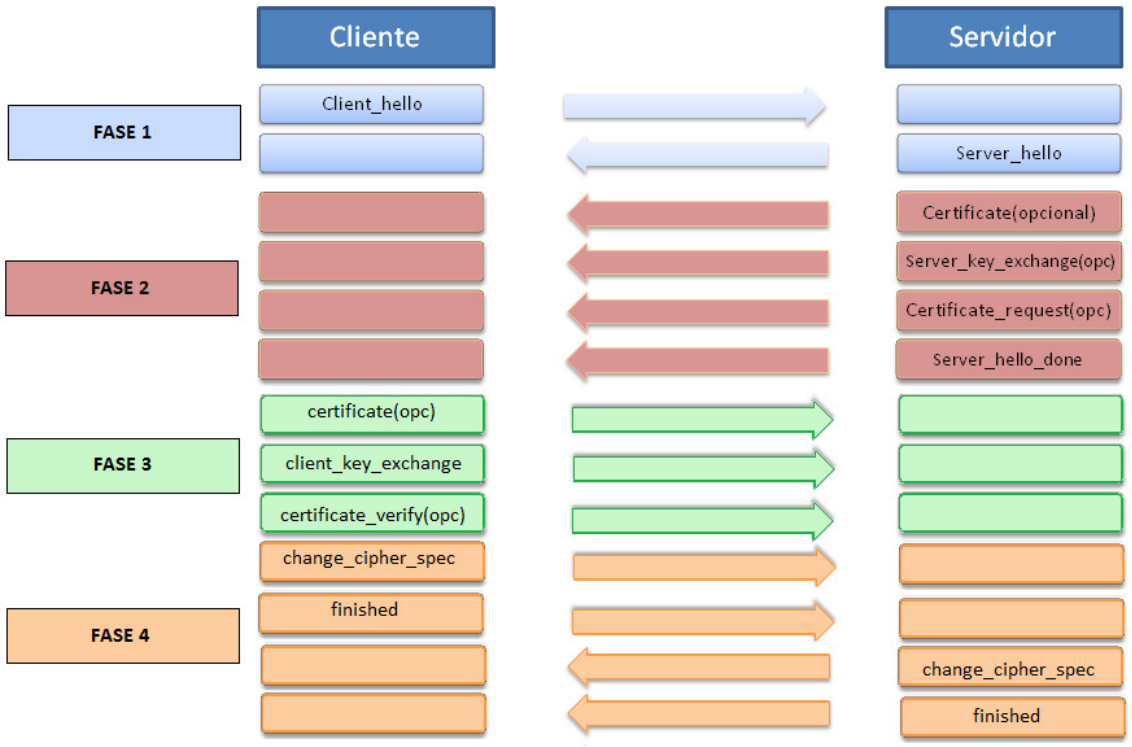

Figura 3.7 Protocolo de Estabelecimento da Comunicação

Na primeira fase são estabelecidas as capacidades de segurança pelos seguintes parâmetros: versão do SSL, valor aleatório para impedir ataques por repetição, ID de sessão para atualizar ou criar uma nova conexão na sessão, o conjunto de cifras suportado e uma lista de métodos de compactação admitida. Na apresentação mútua uma mensagem *client_hello* define os parâmetros da forma que o cliente admite e a mensagem *server_hello* contém os mesmo parâmetros da mensagem *client_hello*, desta forma negociam quais parâmetros irão utilizar.

A segunda fase o servidor pode enviar: o seu certificado (caso seja necessário autenticá-lo), a troca de chaves e solicitar o certificado do cliente. Uma mensagem é enviada sinalizando o término da fase de apresentação mútua.

A terceira fase o cliente envia seu certificado caso tenha sido solicitado, uma mensagem com a troca de mensagens (caso seja o tipo RSA este gera um pré-segredo-mestre aleatório e o criptografa-o com a chave pública fornecida pelo servidor) e uma mensagem para oferecer explicitamente uma verificação do certificado do cliente.

A quarta fase tanto o cliente e o servidor enviam uma mensagem *change_cipher_spec* que faz parte do protocolo *Change Cipher* e não do protocolo de estabelecimento de sessão para determinar o serviço de criptografia, e uma mensagem de conclusão (*finished*), onde esta verifica se o processo de autenticação e troca de chaves foram bem sucedidos. Para garantir a entrega confiável das mensagens é utilizado o TCP.

3.5 Transporte Seguro (Transport Layer Security - TLS)

O TLS 1.0 (RFC2246, 1999) é um protocolo de segurança padronizado para a Internet, desenvolvido pela IETF (Internet Engineering Task Force) baseado na versão 3.0 do SSL (STALLINGS, 2008).

As diferenças entre o SSL 3.0 e o TLS 1.0 são as seguintes:

- Interoperabilidade: o SSL 3.0 e o TLS 1.0 não se comunicam entre si, sendo necessária a escolha de um deles.

- Algoritmos de criptografia de MAC: no TLS utiliza-se o *keyed – Hashing for Message Authentication Code* (HMAC) criptografia mais forte que a utilizada no SSL, que era apenas o MAC.

- Algoritmos para troca de chaves: o TLS suporta todos os algoritmos da SSL 3.0, exceto o Fortezza.

3.6 Considerações Finais

A Internet é um dos grandes veículos de comunicação seja no envio mensagens, arquivos de vídeo, música, texto e outros, além de realizar transações comerciais e financeiras.

Baseado em estatísticas levantadas pelo CERT.br (CERT.br, 2012) incidentes envolvendo ataques a servidores Web, tentativas de fraudes, computadores comprometidos, varreduras e propagação de códigos maliciosos e outros, vem crescendo. Desta forma para garantir que operações comerciais que sejam confidenciais, utiliza-se o protocolo SSL/TLS que garante a autenticidade, confidencialidade e integridade. Sendo que este é conveniente para cada tipo de aplicação, onde a escolha do melhor algoritmo de criptografia e assinatura é garantida (PINHEIRO, VIEIRA e SILVA, 2012).

4 ESTUDO DE CASO: P2PDROID

O P2PDroid é um cliente para a rede P2PLAWS desenvolvido para dispositivos móveis baseados na plataforma Android a partir da versão 2.2. A troca de mensagens é realizada através de sockets em Java, seguindo um protocolo específico para cada tipo de operação (SOUSA, 2011).

4.1 P2PLAWS

A P2PLAWS é uma rede P2P para compartilhamento de recursos aos seus usuários. Ela não depende da tecnologia utilizada para comunicação podendo ser aplicado a serviços Web, sockets ou outros.

4.1.1 Arquitetura

A arquitetura da P2PLAWS (RAMOS, 2009) é baseada na rede Gnutella (Gnutella, 2012), totalmente descentralizada, o serviço de ingresso de um novo nó não é centralizado. Todos os nós da rede possuem as mesmas funcionalidades e operações, desta forma garantindo uma maior tolerância a falhas e robustez (SOUSA, 2011).

A inundação limitada é o mecanismo de busca na rede, onde uma requisição de busca é retransmitida entre os nós até um limite de profundidade estabelecido.

Para que haja o compartilhamento de recursos entre os nós, cinco operações básicas devem ser implementadas pelos clientes: *login*, *connect*, *ping*, *query* e *download.*

4.1.2 Operações Básicas

O P2PLAWS define cinco operações básicas que seus clientes devem realizar são elas:

a) *Login*: É necessário para um nó A ingressar na rede desde que já conheça um nó B participante, e então chamar o método login deste, passando o seu endereço IP ou uma URL para que outros nós possam encontrá-lo. O nó B deve gerar um *PeerID*, que é um identificador único do nó na rede, através do algoritmo SHA-1(*Secure Hash Algorithm*) e enviá-lo junto com a sua tabela de vizinhos(*TabPeers*) ao nó A. Então ao receber a mensagem o nó A torna-se um membro ativo da rede e com acesso aos recursos compartilhados.

b) *Connect*: para atualizar o endereço IP de um nó já participante na rede, deve-se realizar uma chamada do método *connect* para um nó conhecido, informando o seu *PeerID* e o seu novo endereço.

c) *Ping*: o método ping é responsável por atualizar a tabela de nós vizinhos (*TabPeers*) ativo. Funciona enviando uma mensagem de *ping* para todos os seus vizinhos, que por sua vez repassam para seus vizinhos, assim sucessivamente, até o tempo de vida (TTL) se esgote, os nós obtidos destas chamadas são adicionados à *TabPeers* do solicitante, caso haja alguma chamada que não retorne resultado o nó é removido da lista. Este método tem os seguintes parâmetros: *origPeerID*, identificador do nó solicitante, *MessageID*, para que não haja várias respostas a uma única mensagem utiliza-se um identificador único gerado a partir do algoritmo SHA-1, TTL(*Time to live*), tempo de vida ou o número máximo de saltos que o *ping* pode realizar, e *Hops*, a quantidade de saltos realizados até o momento.

d) *Query*: método de busca por recursos na rede. O solicitante envia uma requisição aos seus vizinhos, que por sua vez repassam para seus vizinhos, até atingir o tempo de vida da mensagem, então é retornada uma lista dos recursos que satisfazem a busca, além dos nós onde estes recursos estão disponíveis. Os parâmetros deste método são os mesmos do *ping* mais a palavra-chave usada para a busca.

e) *Download*: método que trabalha junto com a chamada à função *query*. É responsável por solicitar o download de algum dos recursos retornados pela busca.

4.1.3 Metadados

Segundo (SOUSA, 2011), para facilitar o desenvolvimento de clientes para a rede P2PLAWS e padronizar a representação interna dos dados trafegados na rede, foram definidos três arquivos de metadados em XML:

a) MyInfo.xml: detém informações específicas do nó como o seu endereço conhecido na rede, o *PeerID* e um marcador para indicar se é um dispositivo móvel. Como pode ser visto na figura 4.1.

```
<root>
  <myURL>10.0.2.15</myURL>
  <myPeerID>ee29b575b3a07a46a135e9f751d3e24c84600a2a</myPeerID>
  <mobile>true</mobile>
</root>
```

Figura 4.1MyInfo.xml (SOUSA. 2011, p. 30).

b) TabPeers.xml: é a tabela de vizinhos do nó. Internamente é como um conjunto de "MyInfo.xml" dos vizinhos, conforme a figura 4.2.

```
<root>
    <peer>
        <peerID>02171de25809db39303be6fbccb34723abdadb4b</peerID>
        <wsdlURL>10.0.2.15</wsdlURL>
        <mobile>true</mobile>
    </peer>
    <peer>
        <peerID>ee29b575b3a07a46a135e9f751d3e24c84600a2a</peerID>
        <wsdlURL>10.0.2.16</wsdlURL>
        <mobile>true</mobile>
    </peer>
    <peer>
        <peerID>89c1e19f4a60b1b3832dbe7b972da39d319a67b6</peerID>
        <wsdlURL>127.0.0.1</wsdlURL>
        <mobile>false</mobile>
    </peer>
</root>
```

Figura 4.2 TabPeers.xml (SOUSA, 2011, p. 31).

c) TabRecentMessages.xml: os valores dos MessageID de todas as mensagens recebidas pelo nó, exemplo deste arquivo na figura 4.3. Desta forma é possível evitar que um nó

processe uma mensagem repetida.

```
<root>
  <message>M_ID_a85ca4b6d10807111ff9f30b79776e14e510306b</message>
  <message>M_ID_6b066daccc0bea16955c2d6a486e55af65205b65</message>
</root>
```

Figura 4.3 TabRecentMessages.xml (SOUSA, 2011, p. 31).

4.2 Aplicação

A aplicação P2PDroid oferece as operações básicas definidas pela arquitetura da rede P2PLAWS, permitindo o compartilhamento de recursos entre os nós.

Os clientes desta arquitetura realizam a troca de mensagens, comunicação, através de *sockets* seguindo uma sintaxe específica. Segundo (SOUSA, 2011), essas mensagens são divididas em dois tipos, as chamadas de método e retorno. As de métodos são escritas em uma única linha, sendo que é composta pelo nome do método a ser chamado seguido dos seus parâmetros, por exemplo, a chamada do método de *login*, seria assim "login#127.0.0.1". As de retorno são semelhantes às de método, diferem-se por poderem ser compostas por várias linhas, sendo que a última palavra deva ser "FIM", para indicar que o envio foi concluído.

A aplicação funciona da seguinte forma, ao acessá-la pela primeira vez o dispositivo não é considerado um nó da rede, então o usuário é redirecionado para a tela de login, onde terá que informar o endereço de algum nó conhecido ativo na rede, e caso esteja executando em um ambiente de teste deve informar o número da porta que este nó está recebendo conexões, a figura 4.4(A) demonstra estes campos.

A operação sendo concluída com sucesso o nó passa a ser participante na rede e o usuário é redirecionado para a tela inicial (Figura 4.4(B)).

Figura 4.4 (A) Primeiro Login e (B) Tela inicial no ambiente de teste (SOUSA, 2011, p. 34).

A tela inicial possui um campo de texto para que o usuário possa realizar buscas por arquivos, através de palavras-chave. Demonstram-se na tela os resultados e desta forma é possível selecionar um deles para realizar o *download*, como pode ser visto na figura 4.5.

Figura 4.5 (A) Busca do arquivo e (B) resultado da busca (SOUSA, 2011, p. 34).

Após ser escolhido o arquivo a ser baixado, é exibida uma tela com informações adicionais. Como o tamanho do arquivo, nome, e PeerID do nó que o disponibiliza, conforme figura 4.6(A). Verificando estas informações, o usuário pode iniciar o download do arquivo, o qual será salvo na pasta chamada "Downloads" no cartão de memória do seu dispositivo. Os arquivos contidos nesta pasta são automaticamente compartilhados na rede e disponíveis a outros nós. Um exemplo de download de um arquivo pode ser visto na figura 4.6(B).

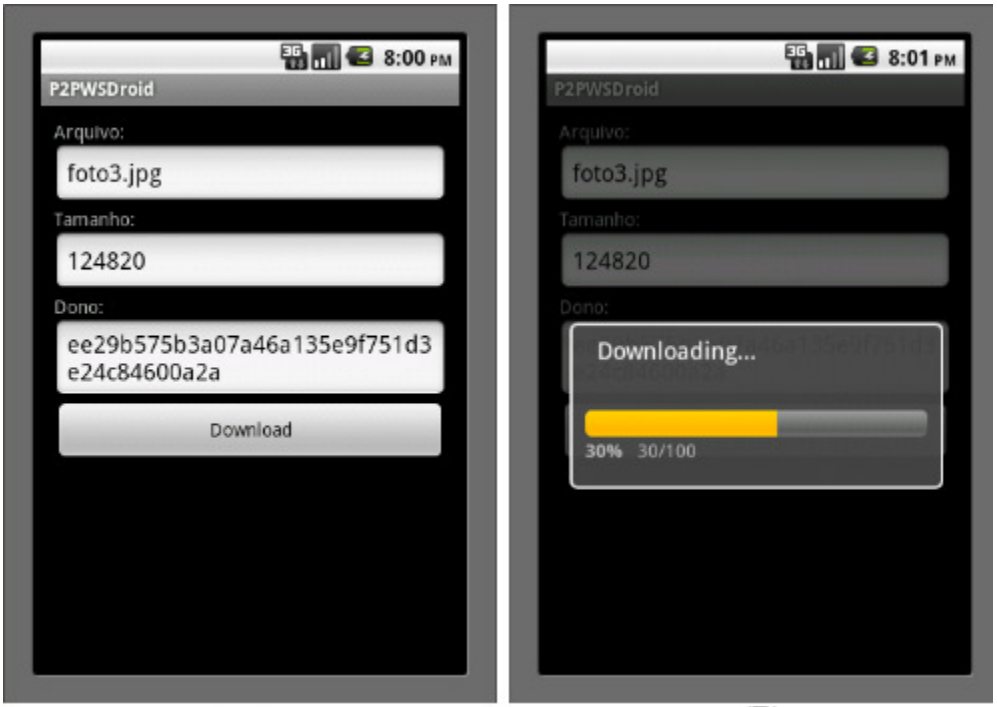

Figura 4.6 (A) Detalhes do arquivo e (B) download do arquivo (SOUSA, 2011, p. 35)

Voltando à tela inicial, através do menu, estão disponíveis outras operações como *ping* e *connect* , além de funcionalidades adicionais, como obter novas credenciais através de um novo *login*, visualizar uma lista de vizinhos e informações do nó, como PeerID, endereço IP e porta.

A funcionalidade novo *login* diferencia-se do realizado pela primeira vez que se executa a aplicação, pelo fato de não ser mais necessário informar o endereço IP de um nó conhecido, basta o usuário selecionar um dos seus vizinhos, que utilizará (Figura 4.7 (A)). Da mesma forma ocorre com o *connect*, só sendo necessário informar para algum dos seus vizinhos as suas novas informações, como visto na figura 4.7 (B). O *ping* funciona como definido na arquitetura P2PLAWS, atualizando a lista de vizinhos do nó e exibindo-a, caso seja selecionado algum, o usuário é redirecionado para uma nova tela onde serão apresentadas as informações deste nó, por exemplo, a figura 4.8.

Figura 4.7 (A) Novo *login* e (B) *connect* (SOUSA, 2011, p. 36).

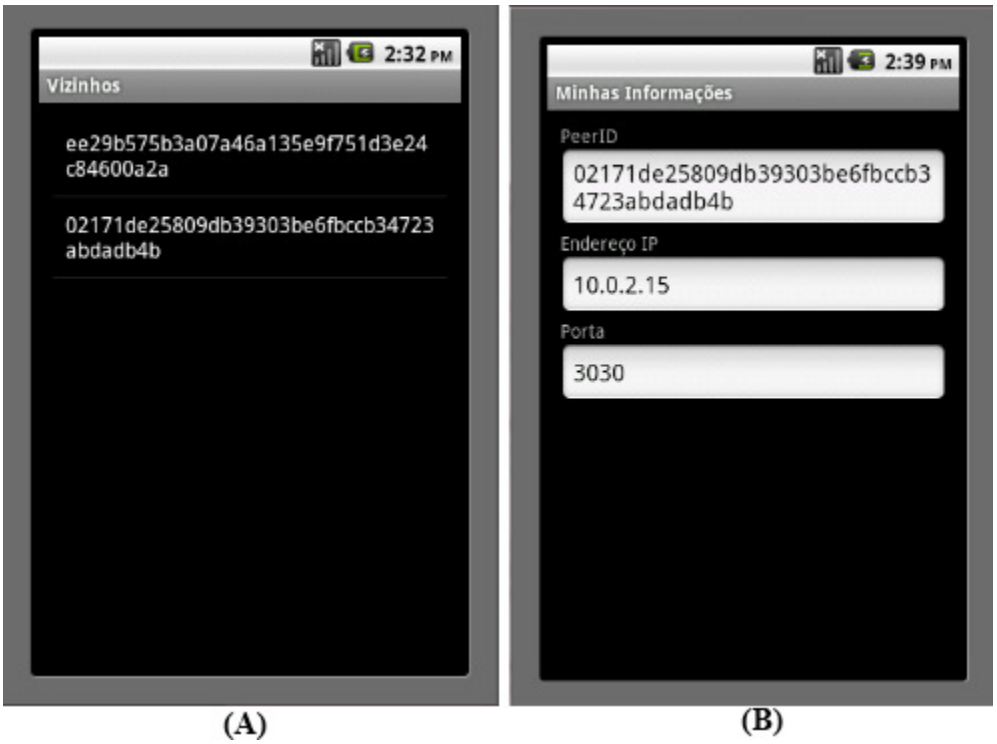

Figura 4.8 (A) Lista de vizinhos e (B) informações do nó (SOUSA, 2011, p. 36).

4.3 Considerações Finais

O cliente P2PDroid oferece o compartilhamento de dados, principalmente arquivos, além das operações básicas definidas pela arquitetura P2PLAWS estarem bem estruturadas e implementadas.

Contudo há necessidade de uma funcionalidade que possa assegurar que os arquivos compartilhados não sofram nenhuma alteração por intrusos, além de garantir ao nó solicitante ter a certeza que determinado arquivo realmente seja do nó que o disponibiliza e manter a confidencialidade do seu conteúdo.

5 IMPLEMENTAÇÃO DE MECANISMOS DE SEGURANÇA EM UMA REDE PEER-TO-PEER PARA DISPOSITIVOS MÓVEIS

Com o grande avanço da Internet e o compartilhamento de dados, principalmente arquivos, via redes P2P houve a necessidade de utilizar um mecanismo que garantisse a sua segurança, visto o grande risco que as informações trafegadas na rede poderiam ser interceptadas ou modificadas por terceiros (JSSE, 2004), além da incerteza de saber se a pessoa com quem está se comunicando é realmente quem diz ser.

Este capítulo demonstra o desenvolvimento e a arquitetura de segurança do cliente P2PDroid baseado na plataforma Android, realizado através das APIs da JSSE (*Java Secure Socket Extension*), que criam sockets seguros baseados em SSL, além de ser utilizado neste trabalho as ferramentas Portecle e o *WireShark*.

5.1 JSSE

O JSSE é um conjunto de APIs que permitem comunicações seguras pela Internet. Fornece um *framework* e uma implementação em Java para versões dos protocolos SSL e TLS, e incluindo funcionalidades para criptografia de dados, autenticação de servidor, integridade de mensagem e opcionalmente a autenticação do cliente (JSSE, 2004). Os desenvolvedores utilizando o JSSE podem permitir a passagem segura de dados entre um cliente e servidor executando qualquer tipo de protocolo de aplicação, tais como *HyperText Transfer Protocol* (HTTP), Telnet ou FTP, através do TCP/IP.

Segundo (JSSE, 2004), suas principais características são:

- Implementado 100% em Java;
- Pode ser exportado para a maioria dos países;
- Fornece suporte para API SSL versão 2.0 e 3.0, e implementação da versão 3.0 do SSL;
- Provê suporte a API e uma implementação para TLS versão 1.0;
- Possuem classes que podem ser instanciadas para criar canais seguros (SSLSocket, SSLServerSocket e SSLEngine);
- Fornece suporte a autenticação do cliente e servidor, que é parte do *hanshaking* normal da SSL;
- Oferece suporte a negociação *cipher suite*.
- Suporte a vários algoritmos de criptografia usados em conjuntos de *cipher suites*;

5.2 Portecle

Portecle é um software livre utilizado para criar, gerenciar e analisar *keystores* (Portecle, 2004),além de chaves, certificados, etc. O software foi desenvolvido por Wayne Grant, Mark Majczyk e Ville Skytta, e mantido pela SourceForge *Project*.

A maioria das funcionalidades de criptografia desta aplicação fora implementado usando a API Bouncy *Castle Crypto*, além de ser uma versão GUI (*Graphical User Interface*) de linha de comando *keytool* fornecido pelo Java SDK

As suas principais características são:

- Criar, carregar, salvar e converter *keystores*.
- Gerar DSA (*Digital Signature Algorithm*) e as entradas de par de chaves RSA auto-assinadas com versão 1 de certificados X.509.
- Importar arquivos de certificados X.509.
- Importar pares de chaves PKCS # 12.
- Clonar e alterar as entradas dos pares de chaves e das *keystores*.
- Verificar os detalhes dos certificados contidos nas *keystores*, arquivos de certificados e conexões SSL/TLS.
- Exportar as entradas de *keystore* em uma variedade de formatos.
- Gerar pedidos de certificação.
- Importar respostas de CA (*Certificate Authority*).
- Eliminar, clonar e renomear entradas de *keystore*.

5.3 Wireshark

WireShark é um analisador de pacotes de rede. Um analisador de pacotes que irá tentar capturar os pacotes que trafegam na rede e mostrar os dados contidos neles de forma mais detalhada possível (WireShark, 2004), além de utilizar o *libpcap* como formato principal para captura de pacotes. O *software* é de código aberto e desenvolvido por uma equipe internacional de especialistas em rede.

Algumas dais principais características que o *WireShark* dispõe:

- Disponível para UNIX, OS X e Windows.

- Captura de dados em tempo real a partir de uma interface de rede.
- Mostra pacotes com informações bastante detalhado.
- Abre e salva pacotes de dados capturados.
- Importa e exporta pacotes de dados de e para vários programas de captura.
- Filtra pacotes em muitos critérios.
- Pesquisa pacotes em muitos critérios.
- Colore pacote exibido com base no filtro.
- Cria várias estatísticas.

5.4 Requisitos do Sistema

Demonstram-se nesta seção os requisitos funcionais e não funcionais do desenvolvimento da segurança do P2PDroid.

Os principais requisitos funcionais são demonstrados no diagrama de casos de uso da figura 5.1.

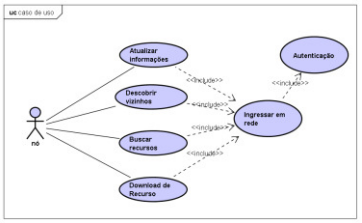

Figura 5.1 Diagrama de Casos de uso.

O desenvolvimento da segurança sustenta-se pelo padrão arquitetural *Layers* (Camadas), visto que o não conhecimento do funcionamento da aplicação não impede a implementação da camada de segurança (SSL). Outros requisitos não funcionais são:

- Sistema executado em um dispositivo móvel baseado na plataforma Android;
- Programação orientada a objeto em linguagem Java;
- Interface ao usuário simples e intuitiva;
- Dispositivo móvel deve está conectado a uma rede;
- A troca de mensagens é realizada através de *sockets*;
- Deve-se ter um certificado digital do cliente e um repositório de certificados dos servidores que o cliente irá aceitar;

5.5 Arquitetura

A arquitetura da aplicação P2PDroid é dividida em 6 componentes, conforme a figura 5.2.

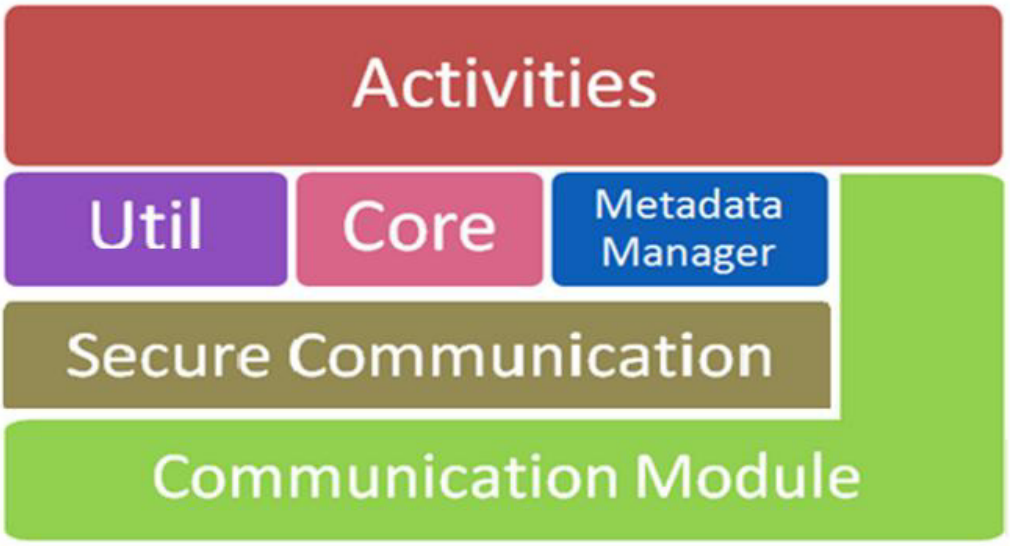

Figura 5.2 Arquitetura do P2PDroid com a camada de segurança implementada.

O componente *Activities* é responsável pela interface da aplicação com o usuário, utilizando todos os componentes menos o *Secure Communication* para tal.

O *Core* tem como função ingressar na rede, atualizar suas informações, procurar vizinhos, buscar recursos e realizar seu *download*, utiliza os componentes Util, *Metadata Manager* e *Communication* Module, para isso.

Util realiza a criação do identificador do nó, além de gerenciar as mensagens que serão enviadas a um dispositivo móvel ou recebidas de outro, identificando quais funções

solicitam repassando ao componente *Communication Module*, para ser posteriormente tratado pelo *Core* da aplicação.

O *Metadata Manager* gerencia os arquivos do formato XML, MyInfo.xml e TabPeers.xml, que respectivamente são responsáveis pelas informações do nó e da lista de seus vizinhos, para isto utiliza os componentes *Core* e *Communication Module*.

Communication Module é responsável pela comunicação com outro dispositivo móvel que possui a mesma aplicação. Este componente possui dois tipos de comportamentos: pode ser cliente ou servidor, dependendo da solicitação que efetuar ou receber, os componentes Util, *Core* e *Metadata Manager*, são utilizados para prover que os dados recebidos através da comunicação possam ser trabalhados no componente Activities.

A *Secure Communication* tem como função criptografar a comunicação, garantindo que qualquer tipo de mensagem de dados enviada ou recebida seja realizado de forma segura. Provendo a integridade dos dados e a autenticação do servidor, desta forma dando suporte ao componente *Communication Module* com seus serviços.

5.5.1 Componente *Secure Communication*

Para o desenvolvimento do componente *Secure Communication* foi utilizado JSSE, protocolo TLS 1.0, liguagem Java, utilizando o Eclipse 3.6 e o SDK, sendo compatível com a versão 2.2 ou superior da plataforma Android, o Portecle para criar os certificados digitais, o cliente Telnet para capturar os pacotes da rede criada entre os dispositivos móveis e o *Wireshark* para analisá-los. Os testes foram realizados junto ao emulador disponível pelo SDK.

Utilizando o Portecle foi criado quatro certificados digitais (gerados pelo algoritmo RSA com chaves de 2048 bits de comprimento), dois no formato .bks (*Bouncy Castle*) e dois .p12 (PKCS12) que o Android suporta. Os arquivos trustore_A.bks e alice.p12, encontram-se na pasta raw do projeto da aplicação P2PDroid que será instalado no dispositivo móvel A, o truststore_A.bks é o repositório de certificados digitais dos servidores que o dispositivo cliente irá aceitar, neste caso do dispositivo B, e o alice.p12 contém o par de chaves, o dispositivo B possui os arquivos truststore_B.bks e bob.p12, com as mesmas funções e instalado da mesma forma como anteriormente citado, com uma única diferença o truststore_B.bks refere-se ao certificado do dispositivo A. A SSL utiliza esses arquivos como

dito na seção 3.5.1, para realizar o estabelecimento da comunicação e criptografia dos dados compartilhados entre os dispositivos móveis na rede.

O cliente P2PDroid por está sendo executado em um ambiente de teste foi necessário realizar algumas alterações junto ao computador, visto que ao ser emulado dois dispositivos móveis, estes possuem o mesmo endereço IP e para diferenciá-los foi atribuído portas como dito na seção 4.3, o endereço do dispositivo móvel emulado passou a ser o endereço IP mais a porta, desta forma para capturar os pacotes na rede criada utiliza-se o cliente Telnet que ao gerar o arquivo no formato *.pcap* é verificado no *Wireshark* .

5.5.2 Detalhes de implementação

Foi utilizado um dos principais pacotes da API JSSE, a javax.net.ssl, conforme pode ser visto suas principais classes dispostas em ordem lógica para a criação de um SSLSocket na figura 5.3.

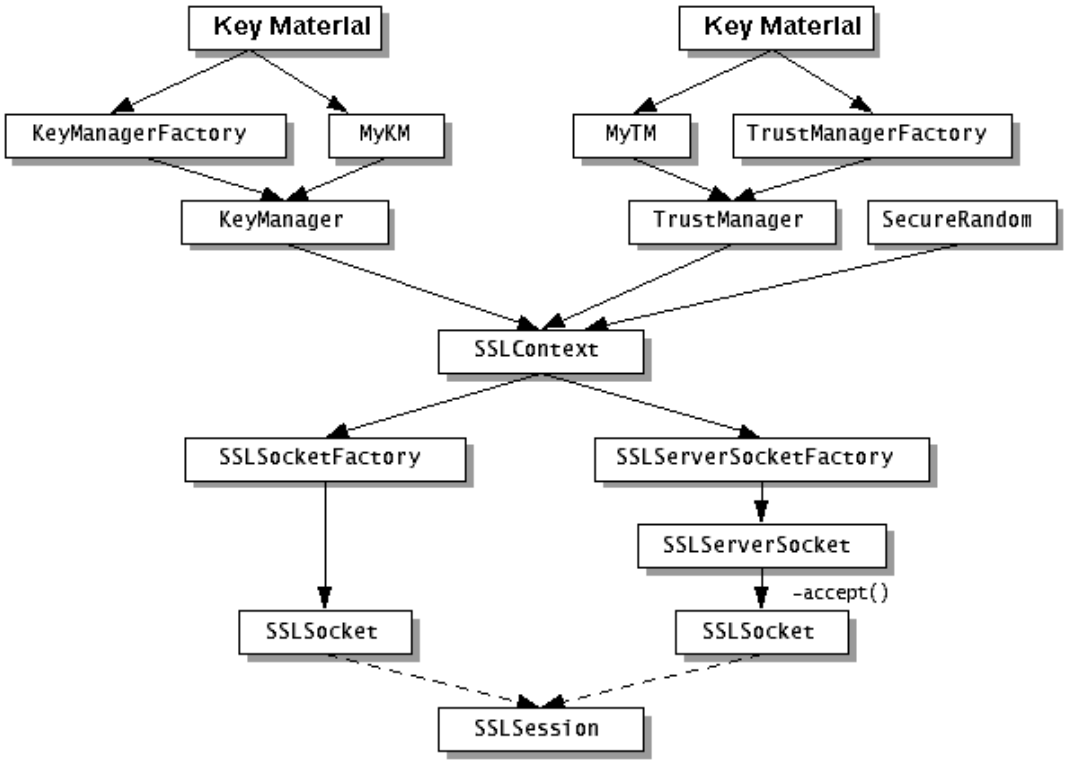

Figura 5.3 Diagrama das principais classes do pacote javax.net.ssl da API JSSE(JSSE, 2004).

Do lado servidor chama-se o método *getInstance*() da SSLContext passando o protocolo TLS para dar mais controle ao contexto criado, e posteriormente é carregado o keystore no KeyManagerFactory dessa forma inicializando-o, além de utilizá-lo para inicializar o SSLContex. O SSLContext cria o SSLServerSocketFactory, conforme a figura 5.4 (A)

```
// Get an SSL context using the TLS protocol
SSLContext sslContext = SSLContext.getInstance("TLS");

// Get a key manager factory using the default algorithm
KeyManagerFactory kmf = KeyManagerFactory
        .getInstance(KeyManagerFactory.getDefaultAlgorithm());

// Load the PKCS12 key chain
KeyStore ks = KeyStore.getInstance("PKCS12");
InputStream keyStoreStream = context.getResources().openRawResource(R.raw.alice);
ks.load(keyStoreStream, "alice".toCharArray());
kmf.init(ks, "alice".toCharArray());

// Initialize the SSL context
sslContext.init(kmf.getKeyManagers(), null, null);

// Create the SSL server socket factory
sssf = sslContext.getServerSocketFactory();
```

Figura 5.4 Parte da codificação do arquivo SocketConnectionThread (A).

O SSLServerSocketFactory é responsável por encapsular a criação e inicializar a configuração de *sockets* seguros do servidor, SSLServerSocket, desta forma cria um *socket* quando for aceita uma conexão, como pode ser visto na figura 5.4 (B).

```
try {
    // Create the secure server socket
    sss = (SSLServerSocket) sssf.createServerSocket(port);
} catch (Exception e) {
    System.out.println("Error: " + e);
    return;
}

try {
    Log.d("SSL", "Server socket factory Waiting for connection");

    System.out.println(">> Thread iniciada, esperando por conexao... ---");

    while (!stoped) {
        // Espera por conexoes
        System.out.println(">> Esperando por conexoes na porta " + port + "...");
         // Wait for an SSL connection
        Socket socket = sss.accept();
```

Figura 5.4 Parte da codificação do arquivo SocketConnectionThread (B).

O cliente configura e inicializa o TrustManagerFactory, que é uma fábrica de

gestores de confiança, carregando o *truststore* sobre o qual irá confiar, como visto na figura 5.5 (A).

```
// Setup truststore
KeyStore trustStore = null;
trustStore = KeyStore.getInstance("BKS");

TrustManagerFactory trustManagerFactory= null;
trustManagerFactory = TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
InputStream trustStoreStream = context.getResources().openRawResource(R.raw.truststore);
trustStore.load(trustStoreStream, "MyPassword".toCharArray());
trustManagerFactory.init(trustStore);
```

Figura 5.5 Parte da codificação do arquivo Client (A).

O SSLContext é instanciado com o protocolo TLS e inicializado com uma lista de gestores de confiança, como na figura 5.5 (B).

```
ssl_ctx = SSLContext.getInstance("TLS");

ssl_ctx.init(null, trustManagerFactory.getTrustManagers(), null);
```

Figura 5.5 Parte da codificação do arquivo Client (B).

O SSLSocketFactory é criado a partir do SSLContext, especificando posteriormente o IP e a porta o qual criará o *socket* para iniciar o *handshake* com o servidor, desta forma estabelecendo a conexão, conforme figura 5.5 (C).

```
SSLSocketFactory socketFactory = (SSLSocketFactory) ssl_ctx.getSocketFactory();
Socket client = (SSLSocket) socketFactory.createSocket(ip, redirPort);


((SSLSocket) client).startHandshake();
```

Figura 5.5 Parte da codificação do arquivo Client (C).

Verifica-se que o *handshake* é realizado autenticando-se o servidor, garantindo a integridade e confidencialidade dos dados que este provê de modo seguro.

5.5.3 Resultados obtidos

Primeiramente foi adicionado o arquivo de texto testep2pdroid, no dispositivo

móvel emulado, Figura 5.6 (A), para que ao ser realizado o download deste arquivo pelo outro dispositivo móvel, Figura 5.6 (B), possa ser capturado os pacotes trafegados na rede criada entre eles pelo cliente Telnet.

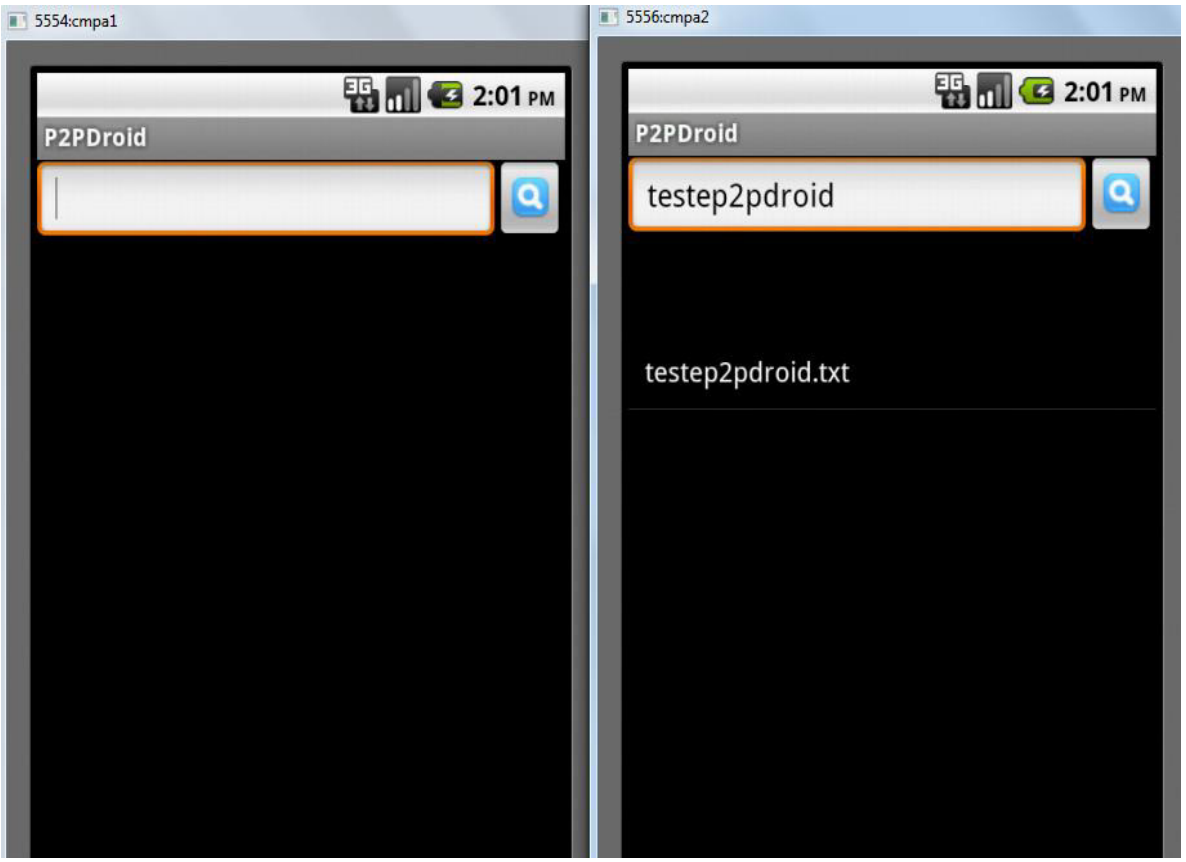

Figura 5.6 Dispositivo móvel emulado (A) e Dispositivo móvel emulado (B).

Para concluir se os dados foram realmente transferidos confidencialmente e de forma íntegra, são capturados os pacotes do mesmo arquivo transferido anteriormente pela aplicação P2PDroid sem a camada de comunicação segura, figura 5.7, e desta forma compara-se os pacotes de cada um verificando com o *WireShark*, constatando ao final que a aplicação implementada com o componente *Secure Communication* provê a criptografia de dados, figura 5.8.

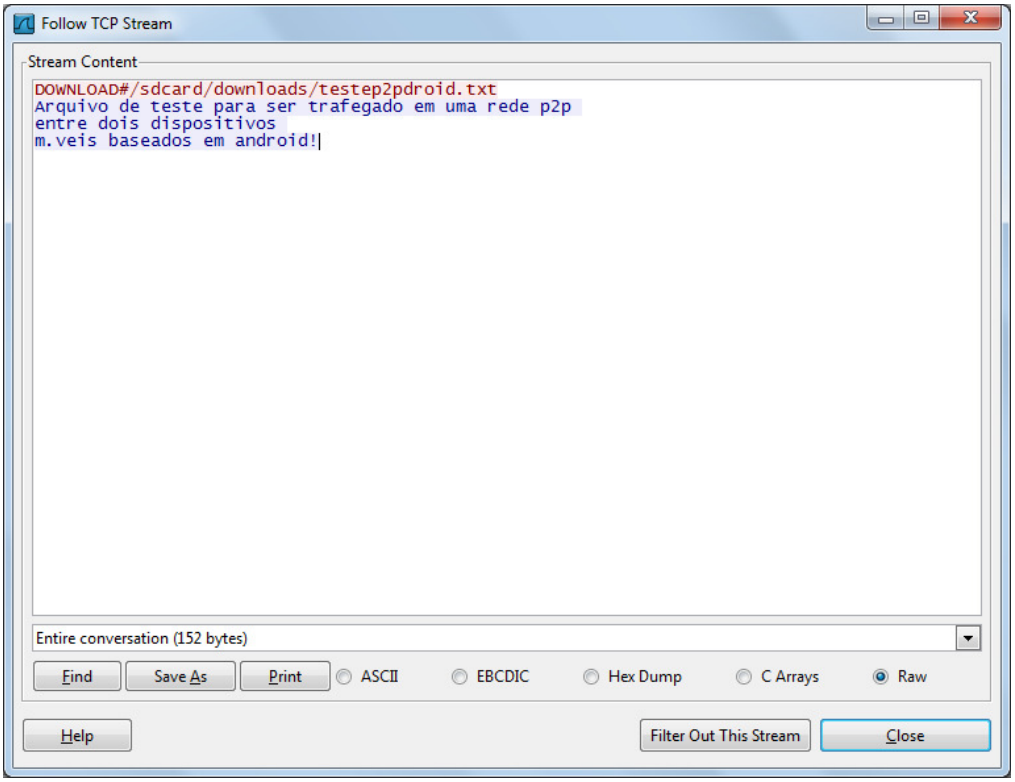

Figura 5.7 Pacote transferido pela aplicação sem o componente *Secure Communication*.

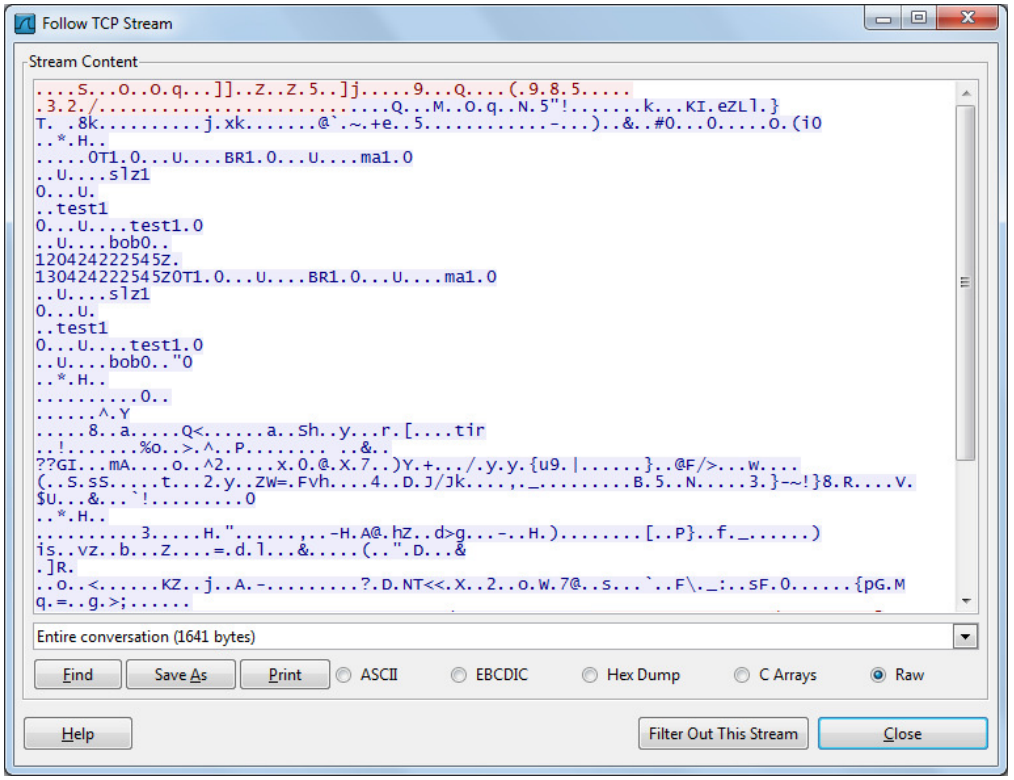

Figura 5.8 Pacote transferido pela aplicação com o componente *Secure Communication*.

6 CONCLUSÃO

A aplicação P2PDroid oferece os serviços de compartilhamento de arquivos a seus usuários baseado na rede P2PLAWS, além de ser desenvolvido para dispositivos móveis com sistema operacional Android de versão 2.2, utilizando a tecnologia de *sockets* como meio de comunicação. Baseado nisto, verifica-se a necessidade de prover uma comunicação segura pelo fato de clientes P2P serem atacados constantemente, para isto foi implementado o componente *Secure Communication*, para desta forma garantir a integridade e confidencialidade dos dados trafegados na rede criada pelos dispositivos móveis.

O componente *Secure Communication* foi implementado com as técnicas de criptografia simétrica e assimétrica, a partir do protocolo TLS versão 1.0 que utiliza as vantagens de cada uma de forma a complementar suas limitações, abordando um estudo do conjunto de API's da JSSE para estabelecer uma maior abstração do desenvolvimento da camada de comunicação segura que fosse suportado pela biblioteca nativa SSL da plataforma Android. Por se tratar de uma implementação para dispositivos móveis encontrou-se dificuldades nos formatos dos keystores e truststores que a plataforma Android suporta a serem criadas no Portecle e da forma de capturar os pacotes trafegados na rede, visto que a rede de um dispositivo móvel emulado trabalha em uma rede virtual atrás de um roteador virtual, tendo que estudar ferramentas para capturar e analisar estes pacotes, que respectivamente são o cliente Telnet e o *WireShark*.

Durante a fase de teste houve problemas em utilizar várias instâncias de emuladores, por demandar uma grande fatia do processamento de memória do computador para cada instância do dispositivo móvel emulado, limitando o teste da aplicação P2PDroid com o *Secure Communication* a dois dispositivos móveis. Com base nos testes realizados podem-se chegar as seguintes conclusões:

- a comunicação é realizada de forma íntegra e confidencial;
- sofreu uma baixa na performance do *download*;
- o estabelecimento da comunicação segura realizada de forma invisível ao usuário;
- a aplicação P2PDroid cliente realiza a autenticação do servidor que disponibilizará o arquivo;

Em trabalhos futuros, pretendem-se definir a inclusão dos arquivos keystores e truststores adicionados na aplicação nas pastas truststore e keystore que já existem no dispositivo móvel baseado na plataforma Android, para que o próprio usuário possa instalá-

los e desta forma utilizá-los na aplicação P2PDroid dando maior robustez a mesma e também solucionar o problema quando clientes não possuem IP's válidos, ou seja, implementar o NAT (*Network Address Translation*).

REFERÊNCIAS

ALVES, Luciano. **Aprenda Passo a Passo a Programar em Android**. [S.l]: Agbook, 2010.

Android Open Source Project. Disponível em < http://source.android.com/ >. Acesso em: 07 maio 2012.

Centro de Estudos, Respostas e Tratamento de Incidentes de Segurança no Brasil (CERT.br). Disponível em <http://www.cert.br/>. Acesso em: 10 abril 2012.

Comitê Gestor da Internet no Brasil (CGI.br). Disponível em <http://www.cgi.br/>. Acesso em: 10 abril 2012

DALVIKVM. Disponível em < http://www.dalvikvm.com/ >. Acesso em: 17 maio 2012.

Extensible Markup Language (XML). Disponível em <www.w3.org/XML/>. Acesso em: 11 maio 2012.

GARFINKEL, Simson; SPAFFORD, Gene. **Comércio & Segurança na Web.** São Paulo: Market Press, 1999.

GARTNER. **Gartner Says Android to Command Nearly Half of Worldwide Smartphone Operating System Market by Year-End 2012**. Disponível em <http://www.gartner.com/it/page.jsp>. Acesso em: 11 maio 2012.

Gnutella. Disponível em <http://gtk-gnutella.sourceforge.net/en/?page=news>. Acesso em: 31 maio 2012.

Google Play. Disponível em <http://play.google.com/intl/pt-BR/about/index.html#utm_ source =HA _ Desktop_BR&utm_medium=SEM&utm_campaign=gplaunch>. Acesso em: 08 maio 2012.

Instituto Nacional de Tecnologia da Informação (ITI). Disponível em http://www.iti.gov.br/twiki/pub/Certificacao/CartilhasCd/brochura01.pdf>. Acesso em: 25 maio 2012.

International Telecommunication Union (ITU). Disponível em < http://www.itu.int/rec/T-REC-X. 509/en >. Acesso em: 16 maio 2012.

Java Secure Socket Extension (JSSE). Disponível em < http://docs.oracle.com/javase/1.4.2/docs/guide/security/jsse/JSSE RefGuide.html>. Acesso em 08 jun. 2012.

KUROSE, James F.; Ross, Keith W. **Redes de computadores e a Internet:** uma bordagem top-down. Pearson Addison Wesley, 2006.

MOZILLA. Disponível em < ttp://www.mozilla.org/projects/security/pki/nss/ssl/draft302.txt >. Acesso em: 17 maio 2012.

National Institute of Standards and Technology (NIST). Disponível em < http://www.nist.gov/index.html >. Acesso em: 15 maio 2012.

National Security Agency (NSA). Disponível em < http://www.nsa.gov/ia/_files/NCES_WSSE_Profile_20080522.pdf >. Acesso em: 16 maio 2012.

PINHEIRO, Fernando Venancio; VIEIRA, Gabriel Serafim; SILVA, Leonardo Gonçalves da. **SSL & TLS**: Redes de Computadores I. Disponível em < http://www.gta.ufrj.br/grad/11_1/tls/index. html >. Acesso em: 21 maio 2012.

Portecle. [S.l]: Ville Skyttä.
Disponível em < http://portecle.sourceforge.net/ >. Acesso em 05 abr. 2012.

RAMOS, Flávio M. P. **Definição de uma Arquitetura P2P Baseada em Reputação e Orientada a Serviços.** Mestrado em Engenharia de Eletricidade, UFMA. 2009.

RFC2246. [S.l]: Dierks & Allen. Disponível em < http://tools.ietf.org/html/rfc2246 >. Acesso em: 22 maio 2012.

Rivest Shamir Adleman (RSA). Laboratories. Cambridge, USA:[s.n]. Disponível em < http://www.rsa.com/rsalabs/ >. Acesso em: 15 maio 2012.

SANTOS, Rogério S. **Plano nacional poderá levar banda larga a 88% da população brasileira**. In: CGI.br (Comitê Gestor da Internet no Brasil). Pesquisa sobre o uso das tecnologias da informação e da comunicação 2009. São Paulo, 2010, pp. 53-57.

Secure Hash Algorithm (SHA-1). Disponível em < http://www.w3.org/TR/1998/REC-DSig-label/SHA1-1_0.html>. Acesso em: 16 maio 2012.

SILVEIRA, Sergio Amadeu da. Novas dimensões da política**:** protocolos e códigos na esfera pública interconectada. **Em Pauta:** revista de Sociologia e Política, vol.17, n. 34, Curitiba, Outubro de 2009.

SOUSA, Lindeberg Barros de. **Redes de Computadores:** guia total. 1.ed. São Paulo: Érica, 2009.

SOUSA, Thiago Nunes de. **Desenvolvimento de uma Rede P2P para a Plataforma Android**. In:_____. Android.[S.l; s.n], 2011.

SOUZA, Adilson Silveira de. **Utilização da tecnologia J2ME para Desenvolvimento de Sistemas de M-Banking.** In:_____. Segurança no J2ME, [S.l; s.n], 2008.

STALLINGS, William. **Criptografia e Segurança de Redes**. 4.ed. São Paulo: Pearson Prentice Hall, 2008.

WireShark: the world´s foremost network protocol analyzer. Disponível em < http://www.wireshark.org/docs/ >. Acesso em: 12 abri. 2012.