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

FELIPE ALEXANDRE OLIVEIRA MACHADO

APLICAÇÕES SEGURAS SOBRE UMA REDE PEER-TO-PEER BASEADA NA PLATAFORMA ANDROID

São Luís
2013

FELIPE ALEXANDRE OLIVEIRA MACHADO

APLICAÇÕES SEGURAS SOBRE UMA REDE PEER-TO-PEER BASEADA NA PLATAFORMA ANDROID

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Mário Antonio Meireles Teixeira

São Luís
2013

Machado, Felipe Alexandre Oliveira.

Aplicações seguras sobre uma rede peer-to-peer baseada na plataforma Android / Felipe Alexandre Oliveira Machado. – São Luís, 2013.

98 f.

Impresso por computador (Fotocópia).

Orientador: Mário Antonio Meireles Teixeira.

Monografia (Graduação) – Universidade Federal do Maranhão, Curso de Ciência da Computação, 2013.

1. Redes P2P 2. Dispositivos móveis 3. Wi-Fi Direct 4. Plataforma Android 5. Criptografia

CDU 004.738

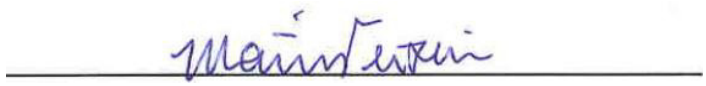
FELIPE ALEXANDRE OLIVEIRA MACHADO

APLICAÇÕES SEGURAS SOBRE UMA REDE PEER-TO-PEER BASEADA NA PLATAFORMA ANDROID

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.

Aprovada em: 10 / 01 / 2014

BANCA EXAMINADORA



Prof. Mário Antonio Meireles Teixeira (Orientador)
Doutor em Informática
Universidade Federal do Maranhão

Geraldo Braz Júnior
Mestre em Informática
Universidade Federal do Maranhão

Maria Auxiliadora Freire
Mestre em Ciência da Engenharia
Universidade Federal do Maranhão

AGRADECIMENTOS

Agradeço primeiramente a Deus por iluminar e guiar minha vida neste desafio. Em segundo lugar, agradeço ao meu orientador Mário Antonio Meireles Teixeira pela paciência, dedicação e conhecimentos passados, e a todos os professores do curso de Ciência da Computação.

À minha família que ofereceu todo o suporte necessário para o meu crescimento pessoal e profissional, assim como esteve ao meu lado nos momentos mais difíceis dessa jornada.

Aos amigos do curso agradeço por todos esses anos de convivência. Foi bastante gratificante aprender, ensinar e produzir com vocês todos os dias.

Por fim, agradeço a todos que direta ou indiretamente contribuíram para a realização deste trabalho.

“Aqueles que se sentem satisfeitos sentam-se e nada fazem. Os insatisfeitos são os únicos benfeitores do mundo”. (Walter S. Landor)

RESUMO

Nas últimas décadas, observou-se um grande crescimento na utilização de redes P2P como ferramenta para compartilhamento de dados. Houve também, a popularização dos dispositivos móveis como *smartphones, tablets* e *notebooks.* Observando esse cenário, a Wi-Fi Alliance criou uma nova especificação chamada Wi-Fi Direct, pelo qual permitiu a união dessas duas tecnologias para a formação de um novo produto, um cliente de uma rede P2P para dispositivos móveis. Dentro deste contexto, a plataforma Android surge como um ambiente favorável ao desenvolvimento de aplicações que desfrutem dessa tecnologia. Neste trabalho é apresentada uma implementação da especificação Wi-Fi Direct, baseada na rede Gnutella e executando sobre a plataforma Android, chamada P2PDroid junto a uma camada de segurança, implementada usando criptografia simétrica e assimétrica, para garantir a segurança e confidencialidade dos dados compartilhados nessa rede.

Palavras-chave: Redes P2P. Dispositivos Móveis. Wi-Fi Direct. Plataforma Android. Criptografia.

ABSTRACT

In recent decades, we observed a large increase in the use of P2P networks as a tool for data sharing. There was also the popularization of mobile devices such as smartphones, tablets and notebooks. Observing this scenario, the Wi-Fi Alliance created a new specification called Wi-Fi Direct, which allowed the marriage of the two technologies to form a new product, a client of a P2P network for mobile devices. Within this context, the Android platform emerges as a favorable development of applications that enjoy this technology environment. In this paper, an implementation of Wi-Fi Direct will be presented. Based on the Gnutella network and running on the Android platform called P2PDroid near a security layer, implemented using symmetric and asymmetric encryption to ensure the security and confidentiality of shared data that network.

Keywords: P2P Networks. Mobile Devices. Wi-Fi Direct. Android Platform. Cryptography.

LISTA DE FIGURAS

Figura 2.1 – Diagrama de alguns servidores e clientes Usenet (Extraído de WIKIPÉDIA, 2012). 16
Figura 2.2 – Logo do ICQ e exemplo das telas (Extraído de ConectadoESPM, 2012). 17
Figura 2.3 – Exemplo de uma Rede P2P centralizada (Extraído de UFRJ, 2008). 19
Figura 2.4 – Ilustração da arquitetura da rede Napster. 20
Figura 2.5 – Exemplo de uma Rede P2P descentralizada (Extraído de UFRJ, 2008). 21
Figura 2.6 – Redes P2P com super-nós (Extraído de UFRJ, 2008). 22
Figura 2.7 – Ilustração da arquitetura de rede Skype. 23
Figura 3.1 – Bugdroid – mascote do Android (Extraído de Google, 2007). 25
Figura 3.2 – Interface do Sistema Operacional Android. 27
Figura 3.3 – Arquitetura Android (Extraído de Android Developers, 2012). 28
Figura 3.4 – Ilustração da interação entre Activities (Extraído de Edureka, 2012). 31
Figura 3.5 – Acesso a dados via Content Provider (Extraído de ABLESON et al, 2009) 32
Figura 3.6 – Ilustração do padrão Observer. 32
Figura 3.7 – Ciclo de vida de uma atividade (Extraído de Android Developers, 2012). 33
Figura 3.8 – O Arquivo AndroidManifest.xml 38
Figura 3.9 – Hierarquia de uma View e Viewgroup (Extraído de Android Developers, 2012). 36
Figura 3.10 – Exemplo de arquivo XML de layout (Extraído de SOUSA, 2011, p. 24). 37
Figura 3.11 – Exemplo de interface desenvolvida utilizando XML (Extraído de SOUSA, 2011). 37
Figura 4.1 – Exemplo de funcionamento da técnica de Código de César (Extraído de WIKIPÉDIA). 40
Figura 4.2 – Exemplo de funcionamento da criptografia simétrica. 41
Figura 4.3 – Exemplo de funcionamento da criptografia assimétrica 42
Figura 4.4 – Exemplo de assinatura digital assimétrica utilizando funções hash. 44
Figura 4.5 – Camadas implementadas pelo SSL. 46
Figura 4.6 – Protocolo de Reconhecimento SSL com Autenticação de Servidor.. 49
Figura 5.1 – Proposta do Wi-Fi Direct. 54
Figura 5.2 – Componentes P2P e topologia. 55
Figura 5.3 – Topologias suportadas pelo Wi-Fi Direct. 56
Figura 5.4 – Tela inicial P2P Droid. 59
Figura 5.5 – Iniciando a descoberta por pares. 60
Figura 5.6 – Lista de pares que estão ao alcance. 60

Figura 5.7 – Informações sobre o par selecionado. 61
Figura 5.8 – Conectando ao par selecionado. 61
Figura 5.9 – Mensagem de convite de outro par. 62
Figura 5.10 – Tela que mostra os dispositivos conectados. 62
Figura 5.11 – Esquema do estabelecimento de comunicação segura do P2PDroid. 66
Figura 5.12 – Arquivo de teste para envio. 68
Figura 5.13 – Captura do pacote sem a camada de segurança 68
Figura 5.14 – Captura do pacote com a camada de segurança. 69

LISTA DE ABREVIATURAS E SIGLAS

3DES	Triple Data Encryption Standard
ADT	Android Development Tool
API	Application Programming Interface
AES	Advanced Encryption Standard
API	Application Programming Interface
DAS	Digital Signature Algorithm
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
HTTP	HyperText Transfer Protocol
IPC	Inter-Process Communication
JSSE	Java Secure Socket Extension
MAC	Código de Autenticação de Mensagens
MD5	Message-Digest Algorithm 5
NIST	National Institute of Standards and Technology
NSA	National Security Agency
OHA	Open Handset Alliance
SDK	Software Development Kit
SHA	Secure Hash Algorithm
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
VPN	Virtual Private Network

SUMÁRIO

1 INTRODUÇÃO 13

1.1 Objetivos 14

1.2 Organização do Trabalho 14

2 REDES P2P 16

2.1 Características 18

2.2 Arquitetura 18

2.2.1 Redes P2P centralizadas 19

2.2.2 Redes P2P descentralizadas 20

2.2.3 Redes P2P com super-nós ou hibridas 21

2.3 Considerações Finais 24

3 ANDROID 25

3.1 Um pouco da história 25

3.2 A plataforma Android 26

3.3 Arquitetura 27

3.4 Desenvolvimento de aplicações 29

3.4.1 Android SDK 29

3.4.2 Componentes dos Aplicativos 30

3.4.3 Ciclo de vida das Activities 33

3.4.4 O arquivo AndroidManifest.xml 35

3.4.5 Interface com o usuário 36

3.5 Considerações Finais 38

4 SEGURANÇA DE REDE 39

4.1 Criptografia 39

4.1.1 Criptografia Simétrica 40

4.1.2 Criptografia Assimétrica 42

4.1.3 Assinatura Digital 43

4.1.4 Funções de Hash 44

4.2 Sockets Seguros (Secure Sockets Layer – SSL) 45

4.2.1 Arquitetura do protocolo SSL 46

4.2.2 Funcionamento do protocolo SSL 47

4.2.3 Transporte Seguro (Transport Layer Security – TLS) ... 49
4.3 JSSE ... 50
4.4 Considerações Finais ... 53
5 REDE P2P DROID SEGURA ... 54
5.1 O Padrão de Comunicação *Wi-Fi Direct* ... 54
5.1.1 Arquitetura ... 55
5.2 Wi-Fi P2P ... 56
5.3 Cliente P2PDroid ... 58
5.3.1 Operações Básicas ... 58
5.3.2 Aplicação ... 59
5.3.3 Processo de Comunicação ... 63
5.4 Camada de Segurança ... 63
5.4.1 Implementação ... 64
5.4.2 Resultados Obtidos ... 68
5.5 Considerações Finais ... 70
6 CONCLUSÃO ... 71
REFERÊNCIAS ... 72
APÊNDICE A – Código-Fonte do aplicativo P2PDroid ... 75

1 INTRODUÇÃO

Antigamente a criptografia era utilizada principalmente na guerra, onde mensagens com determinadas ordens militares eram cifradas no intuito de o inimigo não descobrir o que estava sendo transmitido caso se apoderasse da mesma. O principal objeto da criptografia era dificultar ao máximo a interpretação das mensagens, um dos algoritmos mais antigos de criptografia que se tem notícia é chamado de cifra de César, assim concebido em nome do imperador romano Júlio César, e fora utilizado para esta finalidade (KUROSE, 2006, p.517). Com o surgimento da rede mundial de computadores, a segurança dos dados trafegados tornou-se muito valorizada por instituições, corporações financeiras e grandes empresas de tecnologia, que investem pesado para que seus dados estejam protegidos. Segundo (GARFINKEL e SPAFFORD, 1999) as quatro palavras usadas para descrever todas as diferentes funções criptográficas desempenhadas por sistemas modernos de informação são: confidencialidade[1], autenticação[2], integridade [3]e não-repúdio[4].

Com a popularização dos dispositivos móveis como smartphones e tablets, observou-se a necessidade da criação de aplicações que facilitasse a vida dos usuários. Motivando o surgimento do Android, uma plataforma para dispositivos móveis baseado em Java e executando sobre o núcleo do Linux, de código-fonte aberto, originada por um grupo de empresas da Open Handset Alliance (OHA) liderado pela Google (Android Open Source Project, 2012).

As redes P2P nasceram em consequência do grande aumento de compartilhamento de arquivos pela Internet, principalmente músicas e vídeos. No início da disseminação da Internet e por um longo tempo, o tipo de arquitetura mais utilizada foi a cliente-servidor, porém essa arquitetura apresenta um elevado custo operacional e é suscetível a vários tipos de ataques e falhas, sem contar que, se houver uma indisponibilidade do servidor, os clientes ficam impossibilitados de utilizar os serviços oferecidos. Frente a esse cenário, as redes P2P (*peer-to-peer*) oferecem uma solução descentralizada e auto organizável, onde os clientes, que possuem capacidades e responsabilidade equivalentes, se comunicam diretamente entre si. Este tipo de rede se popularizou com os programas para

[1] Confidencialidade – mantém o conteúdo da informação secreto para todos exceto para as pessoas que tenham acesso à mesma (Fiarresga, 2010,p. 4).
[2] Autenticação de informação – serve para identificar pessoas ou processos com quem se estabelece comunicação (Fiarresga, 2010,p. 4).
[3] Integridade da informação – assegura que não há alteração, intencional ou não, da informação por pessoas não autorizadas (Fiarresga, 2010,p. 4).
[4] Não-repúdio – evita que qualquer das partes envolvidas na comunicação negue o envio ou recepção de uma informação (Fiarresga, 2010,p. 4).

compartilhamento de arquivos, especialmente músicas e hoje são responsáveis por mais da metade do fluxo de dados transmitidos pela Internet. Nesse cenário surge a rede P2PDroid segura, uma arquitetura de rede P2P que provê aos usuários com dispositivos móveis um meio de comunicação e compartilhamento de recursos com segurança. Neste trabalho também é apresentado um cliente para a rede P2PDroid, denominado Cliente P2PDroid. Uma aplicação para dispositivos móveis baseado na plataforma Android que proverá todos os recursos que uma rede P2P totalmente descentralizada oferece.

1.1 Objetivos

Este trabalho tem como objetivo principal implementar mecanismos e aplicações seguras sobre dispositivos móveis executados na plataforma Android, em particular visando a criptografia de dados da rede P2PDroid. Para isto, foi utilizado uma metodologia de criptografia simétrica (STALLINGS, 2008) em conjunto com a criptografia assimétrica (STALLINGS, 2008), garantindo assim, a integridade e velocidade na transmissão dos arquivos.

Os objetivos específicos são listados:

a) Estudo de aspectos de segurança em dispositivos móveis;
b) Implementação de uma arquitetura de segurança para a rede P2PDroid;
c) Implementação de criptografia de dados nos dispositivos conectados à rede P2PDroid.

1.2 Organização do Trabalho

Este capítulo apresentou a motivação e os objetivos de desenvolver um sistema seguro de compartilhamento de arquivos usando a arquitetura P2P e tendo como base a plataforma Android.

O Capítulo 2 apresenta os principais conceitos e características das redes ponto a ponto, bem como os diferentes tipos de arquiteturas e suas principais aplicações.

O Capítulo 3 apresenta plataforma Android como um todo, mostrando os principais conceitos e características desta plataforma, o ambiente de desenvolvimento de aplicações e suas APIs, assim como é estruturada a sua arquitetura,

O Capítulo 4 apresenta os conceitos de segurança de redes e suas formas de criptografia de dados. Além disso, discorre também sobre conexões seguras e transporte seguros.

No Capítulo 5 é definido a arquitetura de uma rede P2P com base na plataforma Android denominada rede P2PDroid. Além disso é mostrado um cliente para esta rede, denominado cliente P2PDroid, implementado com diretivas de segurança e criptografia discutidas no capítulo 4.

O Capítulo 6 traz uma conclusão, resumindo o que foi proposto, propondo melhorias e sugerindo a continuação em trabalhos futuros.

2 REDES P2P

Peer-to-peer (do inglês par-a-par ou simplesmente ponto-a-ponto, com sigla P2P) é uma arquitetura de redes de computadores onde cada um dos pontos ou nós da rede funciona tanto como cliente quanto como servidor, permitindo compartilhamentos de serviços e dados sem a necessidade de um servidor central. As redes P2P podem ser configuradas em casa, em Empresas e ainda na Internet. Todos os pontos da rede devem usar programas compatíveis para ligar-se um ao outro. Uma rede peer-to-peer pode ser usada para compartilhar músicas, vídeos, imagens, dados, enfim qualquer coisa com formato digital (DURANTE, 2004).

Para muitas pessoas as redes p2p nasceram com o surgimento da Napster, porém essa arquitetura sempre existiu, mas não era reconhecido como tal. No século XX, os servidores com endereços de IP fixos ou determináveis podiam sempre se comunicar com outros servidores para usufruir os serviços disponibilizados. Aplicações como, por exemplo, o correio eletrônico, possuíam a capacidade de oferecer um serviço em uma rede distribuída. Entretanto, um aplicativo ganhou destaque na época, a USENET (SCREMIN, HERRERA, PAIXÃO E TAMAKI, 2011).

A USENET surgiu em 1979, e permitia que dois computadores trocassem informação antes da conexão oferecida pela Internet. Esse processo consistia na capacidade de um computador de se ligar a outro, via modem, com a finalidade de pesquisar documentos e transferir documentos para armazenamento local. Nesse modelo não existia nenhuma autoridade central, a distribuição dos documentos era gerada por cada usuário e o conteúdo da rede era replicado para todos os usuários. Um exemplo dessa arquitetura é mostrado na Figura 2.1.

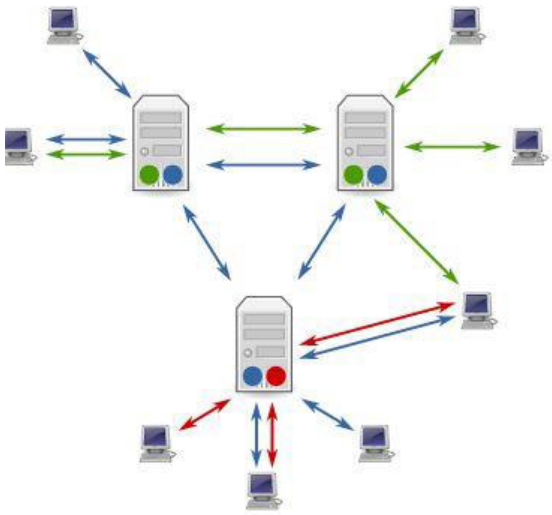


Figura 2.1 - Diagrama de alguns servidores e clientes Usenet (Extraído de WIKIPÉDIA, 2012).

Logo depois surge o ICQ (SCREMIN, HERRERA, PAIXÃO E TAMAKI, 2011), em 1996, Este software oferecia aos seus usuários a capacidade de comunicar-se de forma mais rápida e permitia que seus utilizadores fossem notificados quando seus amigos se conectavam, possibilitava também a troca de arquivos. Apesar de ser classificada como uma aplicação P2P, o ICQ usa uma arquitetura mista entre P2P e cliente/servidor. Seu serviço de notificação de novos usuários é centralizado em um servidor, mas todos os outros serviços são puramente *peer-to-peer*. A Figura 3.2 demonstra algumas telas e o logo dessa aplicação.

Figura 2.2 - Logo do ICQ e exemplo das telas (Extraído de ConectadoESPM, 2012).

Em 1999, surge o Napster (Napster Rhapsody Company, 2013), criado pelo estudante Shawn Fanning, oferecendo aos seus usuários a possibilidade de compartilharem arquivos em *mp3*. Possuía uma arquitetura mista, semelhante ao do ICQ, era suportado por um servidor central que armazenava as listas de músicas das diferentes máquinas e fazia a conexão entre quem buscava um arquivo e quem possuía o arquivo. Era uma aplicação rápida e fácil de usar, porém vulnerável às leis norte-americanas antipirataria, pois violava a Lei de Copyright (Copyright, 2013). Sem o servidor, o Napster não existia, e no início de 2001 a justiça norte-americana conseguiu fechá-lo. Atualmente o Napster vende arquivos de música digital que respeitem o direito autoral.

Por fim, em meados de 2000, surge o Gnutella e a FastTrack que levaram o compartilhamento de arquivos a um passo mais adiante: eliminaram a necessidade de um servidor central para realizar a pesquisa. Essas redes, de fato, ligam os computadores ponto a ponto, tornando-se assim impossíveis de serem fechadas ou barradas por leis.

2.1 Características

A rede P2P é um formato de rede de computadores em que a principal característica é descentralização das funções convencionais de rede, onde o computador de cada usuário conectado acaba por realizar funções de servidor e de cliente ao mesmo tempo.

Seu principal objetivo é a transmissão de arquivos e seu surgimento possibilitou o compartilhamento em massa de músicas e filmes. Com a crescente utilização da rede P2P para este fim, cada vez mais surgem programas para este fim, porém nem sempre eles atendem às expectativas do usuário, pois nem sempre se pode garantir a qualidade e segurança dos arquivos baixados, por se tratar de arquivos de outros usuários e não de fontes confiáveis como produtoras, por exemplo.

Diversas redes operam hoje em dia nestes moldes de compartilhamento, entre elas Kademlia, Gnutela, Kad Network e SoulSeek. Alguns programas valem a pena ser citados quando o assunto é compartilhamento P2P, são eles: SoulSeek, eMule, LimeWire, Ares Galaxy, Shareaza, DreaMule, iMesh e Morpheus.

A ideia da rede P2P é de que seja uma rede totalmente descentralizada. Porém, ao contrário do que se pensa, são raras as redes que possuem uma arquitetura totalmente descentralizada, as chamadas P2P "puras". A maioria das redes P2P possuem um nó centralizador, por mais insignificante que seja, para executar as tarefas típicas de servidores centrais.

Uma ponto que envolve redes P2P é a questão dos "*freeloaders"* - usuários que obtém algum tipo de recurso do sistema P2P, mas não contribui (DURANTE, 2004). Assim, esta questão mostra que redes *peer-to-peer* puras na prática não funcionam como esperado na teoria. Foi baseado nesse contexto que algumas aplicações, como por exemplo, o Napster, encontraram como solução para tal problema, a criação do modelo *peer-to-peer* híbrido, onde os serviços críticos são realizados por servidores, mesclando assim, a arquitetura P2P e cliente/servidor.

2.2 Arquitetura

As redes P2P possuem três tipos de arquiteturas diferentes:

- Redes P2P centralizadas;
- Redes P2P descentralizadas (ditas "puras");
- Redes P2P com uso de super-nós ou híbridas.

2.2.1 Redes P2P centralizadas

As redes P2P centralizadas são caracterizadas pela existência de um servidor central que gerencia os recursos e funcionalidades da rede, estando todos os nós da rede conectados a este servidor (SOUSA, 2011, p. 10). Um exemplo dessa arquitetura é mostrado na Figura 2.3.

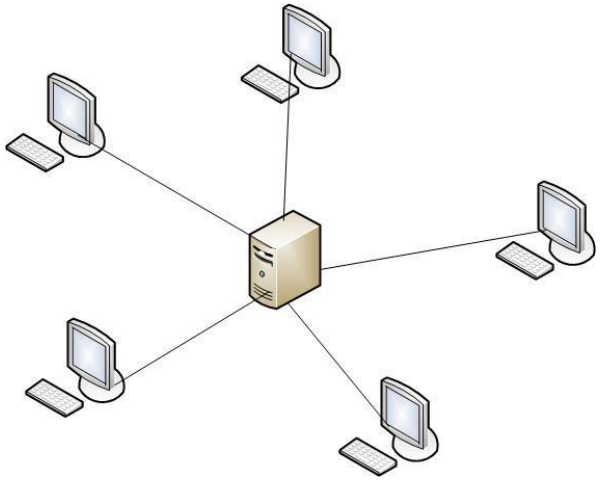


Figura 2.3 - Exemplo de uma Rede P2P centralizada (Extraído de UFRJ, 2008).

São características desse tipo de arquitetura:

- Um servidor central controla as entradas e saídas de *peers* na rede;
- Os *peers* registram no servidor central os recursos que compartilharão na rede;
- Pesquisas por recursos disponíveis nos *peers* são efetuadas pelo servidor central;
- O acesso aos recursos é feito diretamente entre *peers*.

Um exemplo de aplicação que usava essa arquitetura era o Napster. Para funcionar, primeiramente cada computador deveria ter o Napster instalado. Ao abrir o aplicativo, se houver uma conexão com a Internet, o computador se conecta com o servidor central do Napster. O servidor central concentra o diretório de todos os computadores clientes conectados a ele. Se um usuário deseja certo arquivo, ele faz um requerimento ao servidor central do Napster ao qual verifica em seu diretório se existe um arquivo que corresponda ao requerimento. O servidor então envia ao usuário todos os arquivos encontrados (se existirem) juntamente com o endereço IP, nome do usuário, tamanho do arquivo, tempo de ping e taxa de transferência de cada um. O usuário escolhe o arquivo desejado da lista correspondente e

tenta estabelecer uma comunicação direta com o local onde se localiza o arquivo escolhido. Se a conexão for estabelecida, então o computador cliente onde se localiza o arquivo é considerado o "servidor". Assim que o "servidor" conclui a transferência do arquivo para o usuário, a conexão é interrompida. A Figura 2.4 demonstra a arquitetura da rede Napster.

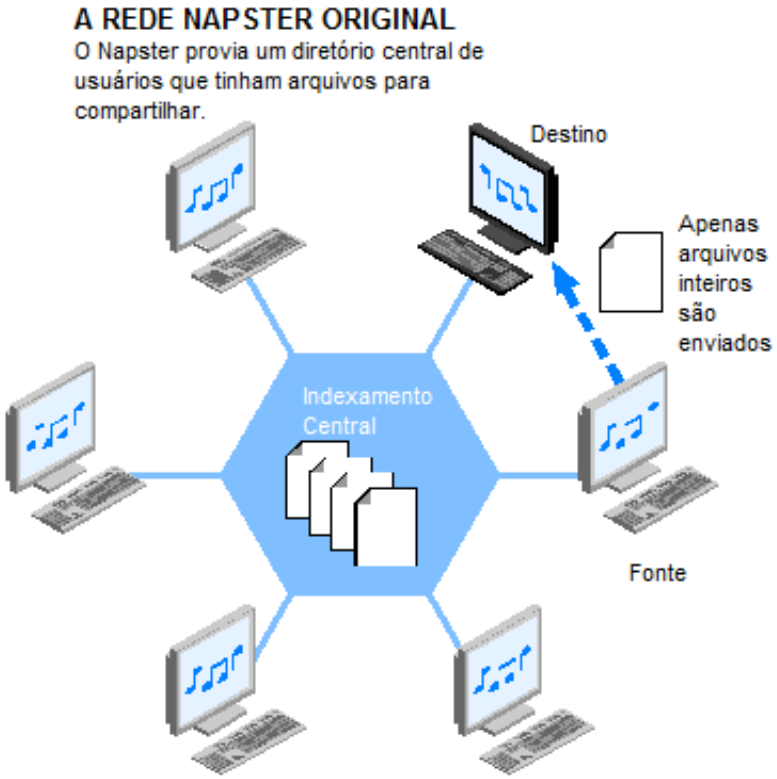


Figura 2.4 - Ilustração da arquitetura da rede Napster.

2.2.2 Redes P2P descentralizadas

Nas redes P2P descentralizadas, a total liberdade é a principal característica. Nessa arquitetura não existe um servidor central (é dita P2P "pura") ou qualquer tipo de hierarquia dos nós. Cada nó pode estar ligado com qualquer outro nó da rede (SOUSA, 2011, p. 10). A Figura 2.5 demonstra um exemplo dessa arquitetura.

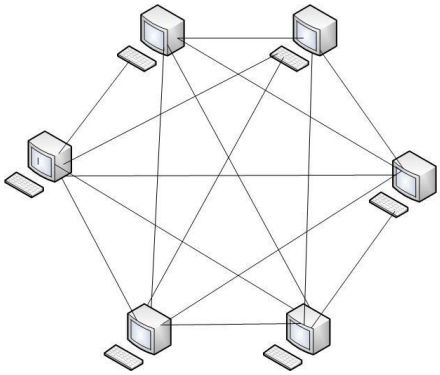


Figura 2.5 - Exemplo de uma Rede P2P descentralizada (Extraído de UFRJ, 2008).

São características desse tipo de arquitetura:

- Não há um elemento central;
- Todos os nós possuem papel equivalente;
- As pesquisas por recursos compartilhados são feitas por inundação;
- Gera um alto tráfego na rede;
- Desempenho das pesquisas é ruim devido à necessidade de contactar muitos nós e aguardar a resposta.

O Freenet (Freenet, 2013) é um exemplo de sistema que usa esse tipo de arquitetura. Freenet é um sistema P2P completamente descentralizado e distribuído. Toda a comunicação é coordenada pelos pares em nível global. Ele opera como um sistema de arquivos distribuídos independentemente da localização entre vários computadores que permitem que a inserção, armazenamento e requisição de dados aconteça de forma anônima.

Um nó é um simples computador que "percorre" todo o sistema, e todos os nós são tratados como iguais pela rede. Cada nó mantém sua própria unidade de armazenamento a qual é disponibilizada para a rede tanto para a leitura e escrita quanto para a tabela de roteamento dinâmico que contém os endereços de outros nós. Isso faz com que qualquer ponto de falha ou controle seja excluído. O protocolo do Freenet faz com que seus nós se organizem espontaneamente dentro de uma rede eficiente.

2.2.3 Redes P2P com super-nós ou híbridas

Nas redes P2P com super-nós, a principal característica é a existência de nós na rede com atribuições de servidor. Estes nós são chamados de super-nós. Eles gerenciam sub-redes de nós e se comunicam com os demais super-nós da rede, que formam uma rede entre si

mesmos (SOUSA, 2011, p. 11). Sendo assim, este tipo de arquitetura funciona como um hibrido entre a arquitetura centralizada e a descentralizada. A Figura 2.6 demonstra um exemplo dessa arquitetura.

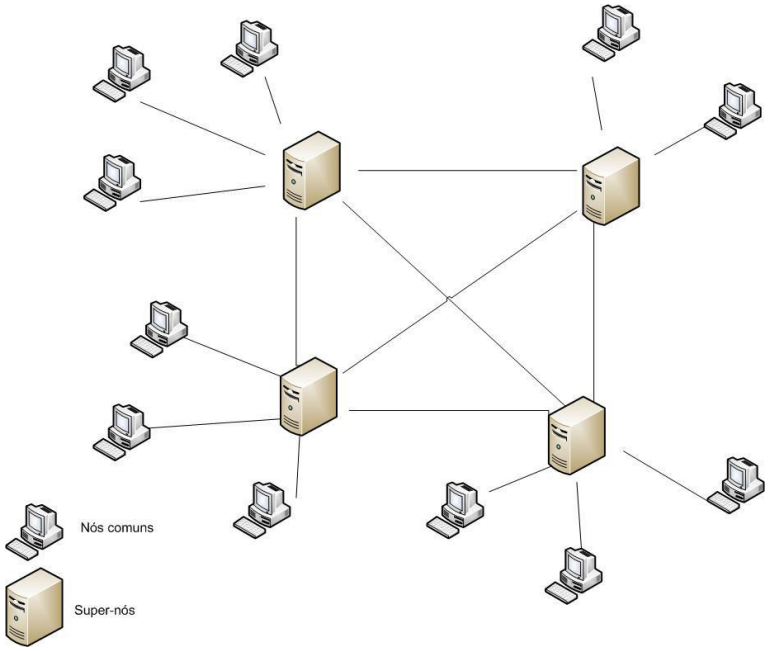


Figura 2.6 – Exemplo de uma Rede P2P com super-nós (Extraído de UFRJ, 2008).

São características desse tipo de arquitetura:

- Super-nós permitem o ingresso dos nós na rede, podendo também exercer atividades de coordenação do funcionamento da rede, indexar os recursos compartilhados pelos nós e permitir a busca por estes recursos;
- Após localizado, um recurso pode ser obtido a partir da interação direta entre os nós;
- Falha de um super-nó pode ser tolerada elegendo dinamicamente outro super-nó.

Um exemplo de uma aplicação bem famosa que usa essa arquitetura é o Skype (Skype, 2013). O Skype é um aplicativo que permite a comunicação por áudio ou vídeo entre usuários. Utiliza super-nós para busca e indexação de conteúdo e um servidor de *login* para autenticação. O servidor de *login* é uma aplicação à parte e constitui o único elemento central na rede, mantém informação de *login* e senha dos usuários da rede. Todos os clientes dessa rede rodam localmente a aplicação e são chamados de clientes, onde alguns deles podem atuar como super-nós.

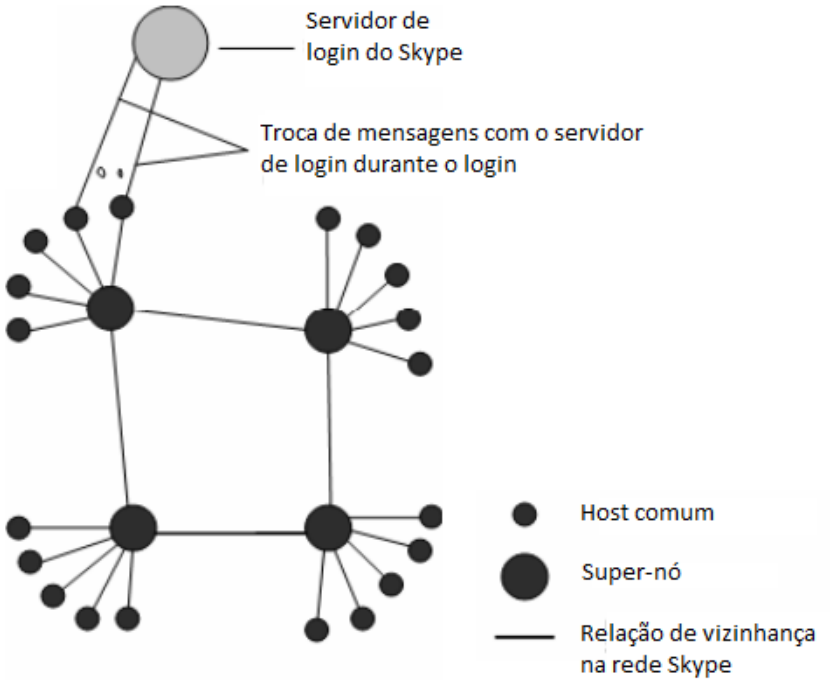


Figura 2.7 - Ilustração da arquitetura de rede Skype.

O processo de conexão é simples; a aplicação cliente efetua o *login* na rede se registrando no servidor de *login*, cujo endereço foi informado por outros hosts pertencentes à rede. Após o *login* cada cliente deve construir e manter atualizada uma tabela para identificar os super-nós conhecidos, esta tabela é chamada de host cache e contém o endereço IP e o respectivo número da porta dos super-nós que estiverem on-line. A aplicação também mantém outra listagem local criptografada armazenada no registro do sistema operacional, que é a lista de contatos. Pode-se observar que o Skype adota uma estratégia de utilizar o máximo possível o hardware do usuário, o qual percebe-se que a rede só é responsável pelas informações de *login*, ficando a cargo dos nós todo o processamento e armazenamento das informações.

2.3 Considerações Finais

Com o constante crescimento da rede mundial de computadores e facilidade de acesso a mesma, as redes P2P crescem cada vez mais e se tornam mais populares. Novos programas, novas tecnologias e novas redes surgem a cada ano, otimizando a utilização das redes P2P e atraindo cada vez mais novos usuários e até empresas que passam a utilizar o P2P.

No outro lado da ponta temos as indústrias cinematográficas, fonográficas e as desenvolvedoras de softwares que se sentem prejudicadas com seus produtos sendo distribuídos gratuitamente pelas redes P2P, violando regras de direitos de cópia. Como o modo de funcionamento dessas redes dificulta o combate legal de violação dos direitos, as empresas estão cada vez mais dispostas a afastar usuários das redes P2P, contaminando arquivos na rede e até mesmo atacando diretamente com ataques de negação de serviço.

3 ANDROID

Android é um sistema operacional baseado no núcleo do Linux para dispositivos móveis (AROUCHA, 2012). Desenvolvido pela Open Handset Alliance (OHA, 2011), liderada pelo Google e outras empresas. Segundo a Google, mais de 1 milhão e 300 mil aparelhos com este sistema operacional são ativados todos os dias, utilizado por vários fabricantes de celulares como: HTC, Samsung, Sony, Motorola, LG e recentemente a Positivo Informática. Sua versão mais recente é a 4.4, chamada de KitKat. Sua última versão foi lançada em 31 de outubro de 2013 junto com o novo smartphone do Google, o Nexus5. O mascote oficial do Android é chamado de Bugdroid, como mostra a Figura 3.1.

Figura 3.1 - Bugdroid – mascote do Android (Extraído de Google, 2007).

Seu grande diferencial é que, como o código é liberado, programadores de todo o mundo criam aplicativos, ou seja, pequenos programas para rodar nos dispositivos que tenham o sistema. São aplicativos que melhoram as funções dos dispositivos móveis, expandido suas funcionalidades. São diversos aplicativos que auxiliam no dia a dia dos usuários, dentre eles pode-se citar os leitores de e-books, tocadores de mp3, bússolas digitais e, claro, jogos digitais. Esses aplicativos são todos catalogados e ficam à disposição dos usuários na Play Store (Google Play, 2013), uma loja virtual acessível via celular, tablets e computadores; muitos deles gratuitos.

3.1 Um pouco da história

Em Outubro de 2003 era fundada a Android, Inc., em Palo Alto, Califórnia por Andy Rubin, Rich Miner, Nick Sears e Chris White. Isso distorce um pouco o fato que muitos acreditam, de que a Google fundou esta empresa, ela na verdade comprou-a em julho de 2005. A Android Inc. era uma pequena empresa em Palo Alto, California, USA. No tempo

que a Google adquiriu a Android Inc., conduzido por Andy Rubin, foi desenvolvida uma plataforma de telefone móvel baseado em Linux, com o objetivo de ser uma plataforma flexível, aberta e de fácil migração para os fabricantes. Em Dezembro de 2006 surgiram mais especulações de que a Google estava entrando no mercado de telefones móveis.

No ano de 2007, foi criado um consórcio chamado Open Handset Alliance formado por 47 empresas e liderado pelo Google. A função desta reunião de empresas era criar um novo sistema operacional para celulares que fosse um sistema aberto, ou seja, sem que fabricantes tivessem que pagar licença de uso. Foi nesse momento que nasceu o Sistema Operacional Android.

O HTC Dream foi o primeiro telefone comercialmente disponível a rodar no sistema Android, lançado em 22 de outubro de 2008. No dia 27 de maio de 2010 Matias Duarte, antigo diretor de interface do WebOS (sistema operacional da Palm), junta-se à equipe do Android após a compra da Palm pela HP.

3.2 A plataforma Android

A plataforma Android tem como base o *kernel* do Linux 2.6, onde foram feitas várias alterações para que o sistema ficasse otimizado e adaptado aos dispositivos portáteis e é responsável pelas tarefas fundamentais do sistema tais como, gerenciamento de discos, gerenciamento de processos, segurança, entre outras funções básicas de um sistema operacional.

A plataforma Android possui diversas ferramentas que auxiliam no desenvolvimento de poderosas aplicações, dentre elas estão:

- *Application Framework* – permite o reuso e a substituição de componentes;
- Máquina Virtual *Dalvik* – roda os softwares escritos na linguagem Java e compilados no formato de *bytecodes*;
- Navegador baseado no *Webkit Engine* – também usado pelo Safári;
- Ambiente gráfico;
- Gerenciador de pacotes;
- Biblioteca 2D e 3D;
- Banco de dados SQLite – armazena os dados estruturados;
- Suporte para mídias: áudio, vídeo, formato de imagens (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG E GIF);

- Redes sem fio – telefone GSM, *bluetooth*, EDGE, 3G, 4G, Wi-Fi, Wi-Fi Direct, GPS e compasso.

A Figura 3.2 ilustra a interface com o usuário do Sistema Operacional Android.

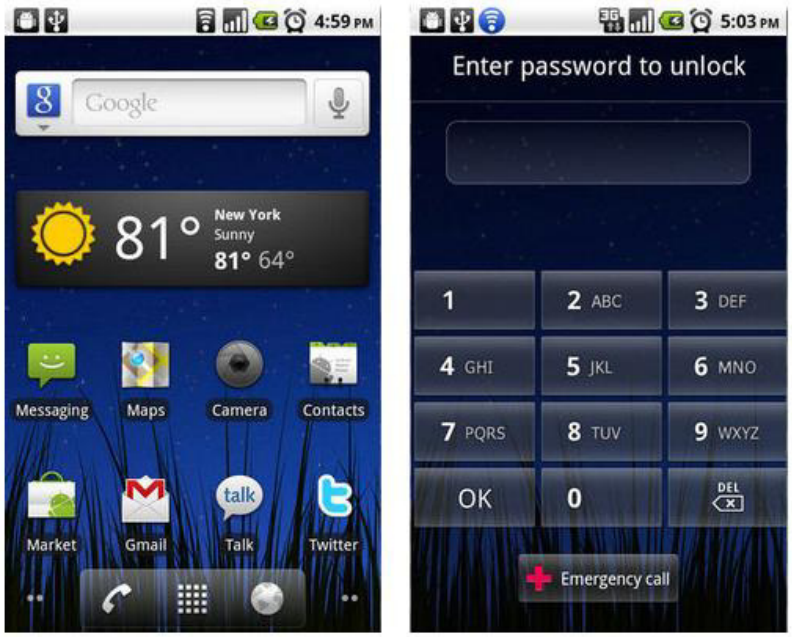


Figura 3.2 - Interface do Sistema Operacional Android.

3.3 Arquitetura

A arquitetura do Android é dividida em cinco camadas, são elas: *kernel* Linux, bibliotecas, ambiente de execução, *framework* e as aplicações. Na Figura 3.3 é demonstrado os componentes que compõe cada uma dessas camadas.

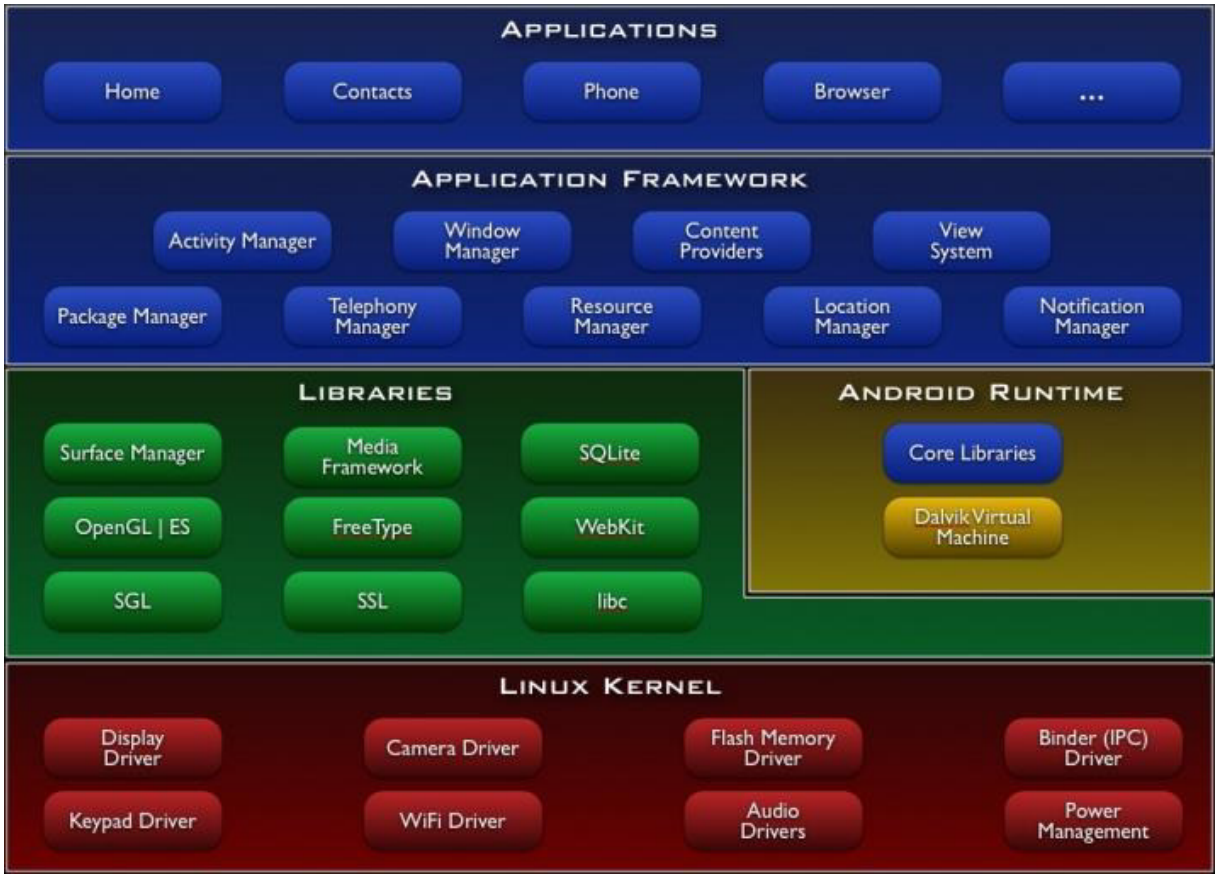


Figura 3.3 - Arquitetura Android (Extraído de Android Developers, 2012).

O sistema operacional Android geralmente é referenciado pelo Google como uma pilha de softwares. Cada camada dessa pilha agrupa vários programas que suportam funções específicas do sistema operacional;

A camada mais baixa da pilha é o ***kernel***. Como dito anteriormente, foi usado a versão 2.6 do Linux para construir o *kernel* do Android, que inclui os programas de gerenciamento de memória, configurações de segurança, software de gerenciamento de energia e vários drivers de hardware.

Na próxima camada encontram-se as **bibliotecas**. Escritas em C/C++, usadas por diversos componentes do sistema Android (AROUCHA, 2012). Por exemplo, a biblioteca do framework de mídia suporta a reprodução e a gravação de vários formatos de áudio, vídeo e imagem.

A próxima camada, que encontra-se no mesmo nível da anterior, é o ambiente de tempo de execução (**Android *Runtime***) que inclui um conjunto de bibliotecas do núcleo Java onde fornece a maioria das funcionalidades disponíveis na linguagem, 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 (AROUCHA, 2012).

A penúltima camada é o ***framework* de aplicação.** Inclui os programas que gerenciam as funções básicas do telefone, como alocação de recursos, aplicações de telefone, mudança entre processos ou programas, etc. Os desenvolvedores de aplicações para Android têm acesso total a esse framework. Isso possibilita que eles tirem vantagem das capacidades de processamento do Android e suportem recursos quando estão construindo uma aplicação Android.

Na última camada estão as aplicações em si que são escritas em linguagem Java. É onde encontram-se as funções básicas do dispositivo, como fazer chamadas telefônicas, acessar o navegador Web ou acessar sua lista de contatos. Essa é a camada que usuários comuns mais usam. Apenas os programadores do Google, os desenvolvedores de aplicação e os fabricantes de hardware acessam outras camadas inferiores da pilha.

3.4 Desenvolvimento de aplicações

Para começar a construir aplicações Android, o desenvolvedor tem de estar familiarizado com a linguagem de programação Java. Assumindo que o desenvolvedor conheça bem a linguagem Java, o próximo passo é fazer o download do kit do desenvolvedor Android, ou Android SDK (Android Developers, 2013). O SDK dá ao desenvolvedor acesso à interface de programação do Android (API). Todos os arquivos e códigos escritos são compactados em um único pacote de formato *apk*, depois de compilados, e são utilizados pelo Android para instalação da aplicação.

3.4.1 Android SDK

SDK, é a sigla de *Software Development Kit*, ou seja, Kit de Desenvolvimento de Software ou Kit de Desenvolvimento de Aplicativos. São disponibilizados por empresas ou projetos *open source* para que programadores externos tenham uma melhor integração com o software proposto.

O SDK do Android inclui várias ferramentas como exemplos de aplicações, emulador de telefone, ferramentas para *debugging*, memória e análise de performance. O emulador do Android é um programa que replica as funções de um telefone executando na plataforma Android e pode ser utilizado pelos desenvolvedores com a finalidade de testarem suas aplicações como se estivesse executando em um telefone físico.

A IDE (Integrated Development EnvironmentI) recomendada para desenvolvedores de aplicativos Android é o Eclipse (Eclipse, 2012), com o *plugin* ADT (Android Development Tool). O ADT amplia os recursos auxiliando a criar rapidamente novos projetos Android, 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; oferece também uma interface gráfica com usuário integrada com um editor XML (Extensible Markup Language) personalizado para a criação das telas dos aplicativos.

3.4.2 Componentes dos Aplicativos

Uma aplicação Android é orientada a eventos. Um evento é um sinal, que pode ter sido originado pelo usuário, como um toque na tela, ou não. Logo, uma aplicação é dividida em diversos tipos de componentes, que compartilham um nome de pacote e de aplicação em comum.

O Google divide todas as suas aplicações em quatro blocos básicos de construção (nem todas as aplicações fazem uso de todos os blocos de construção):

- *Activities* – atividades ou telas
- *Services* – serviços em background
- *Content Providers* – disponibilizam dados
- *Broadcast Receivers* – respondem a avisos gerais do sistema

Os tipos de componentes são declarados e implementados na camada *framework* e o desenvolvedor os estende para criar seus componentes específicos.

Uma *Activity* é um componente da aplicação que fornece uma tela com o qual os usuários podem interagir para fazer algo, como discar o telefone, tirar uma foto, enviar um e-mail ou ver um mapa. Um aplicativo normalmente consiste de várias *Activities*. Normalmente, uma *Activity* em um aplicativo é especificada como a *Activity* principal ou "*Main Activity*" e é apresentada ao usuário quando se inicia o aplicativo pela primeira vez. Cada *Activity* pode chamar outra *Activity* a fim de executar ações diferentes e essa chamada é guiada por meio dos *Intents*. Os *Intents* especificam tanto a *Activity* que se deseja iniciar ou descrevem o tipo de ação que será realizada, além de transportar dados entre as *Activities*. Estas *Activities* seguem um ciclo de vida que será apresentado na seção 3.5.3. Um exemplo de interação entre *Activities* é ilustrado na Figura 3.4.

Figura 3.4 - Ilustração da interação entre Activities (Extraído de Edureka, 2012).

Um *Service* é um componente da aplicação que pode executar operações de longa duração em segundo plano e não fornece uma interface de usuário. Um componente de aplicativo pode iniciar um serviço que irá continuar a executar em segundo plano, mesmo se o usuário alternar para outro aplicativo. Além disso, um componente pode vincular a um serviço para interagir com ele e até mesmo realizar a comunicação entre processos (em ingês Inter-Process Communication ou IPC). Por exemplo, um serviço pode manipular transações de rede, tocar música, executar arquivo I/O ou interagir com um provedor de conteúdo, tudo em segundo plano.

Os *Contents Providers*, ou provedores de conteúdo, gerenciam o acesso a um conjunto de dados estruturados. Eles encapsulam os dados e fornecem mecanismos para a definição de segurança de dados. O *Content Provider* é a interface padrão que conecta os dados em um processo com código sendo executado em outro processo, por exemplo, se o desenvolvedor necessitar acessar as informações dos contatos do telefone, ele pode fazer isso usando o *Content Provider* específico para isso. A Figura 3.5 ilustra o a forma de como o *Content Provider* pode se comunicar com outras aplicações.

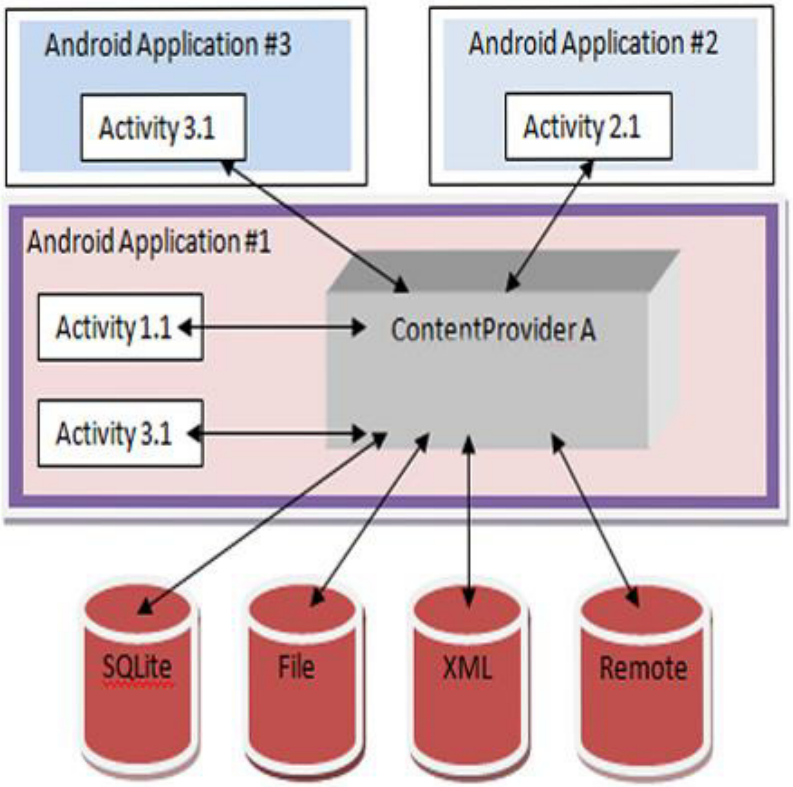


Figura 3.5 - Acesso a dados via *Content Provider* (ABLESON et al, 2009).

Broadcast Receiver é o componente que responde a alertas gerais do sistema. Podem ser gerados pelas *activities* ou não. Por exemplo, o *Intent* SCREEN_OFF é gerado pelo sistema quando a tela é bloqueada e um *Broadcast Receiver* pode responder a esse alerta quando o mesmo for capturado pela aplicação. Você pode tomar atitudes de acordo com o tipo de alerta recebido. Um *Broadcast Receiver* segue o padrão *Observer* (Figura 3.6). Ele se registra e fica observando se determinado evento geral acontece. É diferente da *Activity*, que somente observa eventos gerados pelas suas *Views*. Este tipo de componente pode receber eventos gerados pelo código de sua aplicação, de outras aplicações ou mesmo do sistema Android.

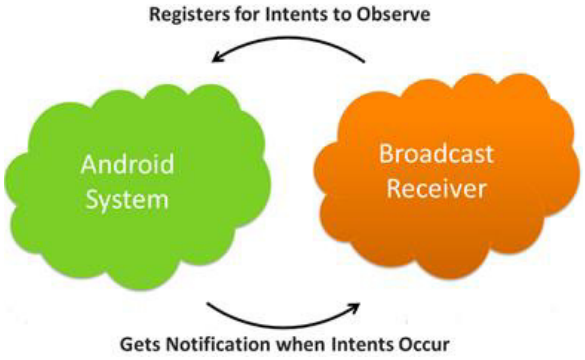


Figura 3.6 - Ilustração do padrão *Observer*.

3.4.3 Ciclo de vida das Activities

Uma das coisas que é mais importante conhecer sobre a *Activity* é o seu ciclo de vida, que é demonstrado na Figura 3.7.

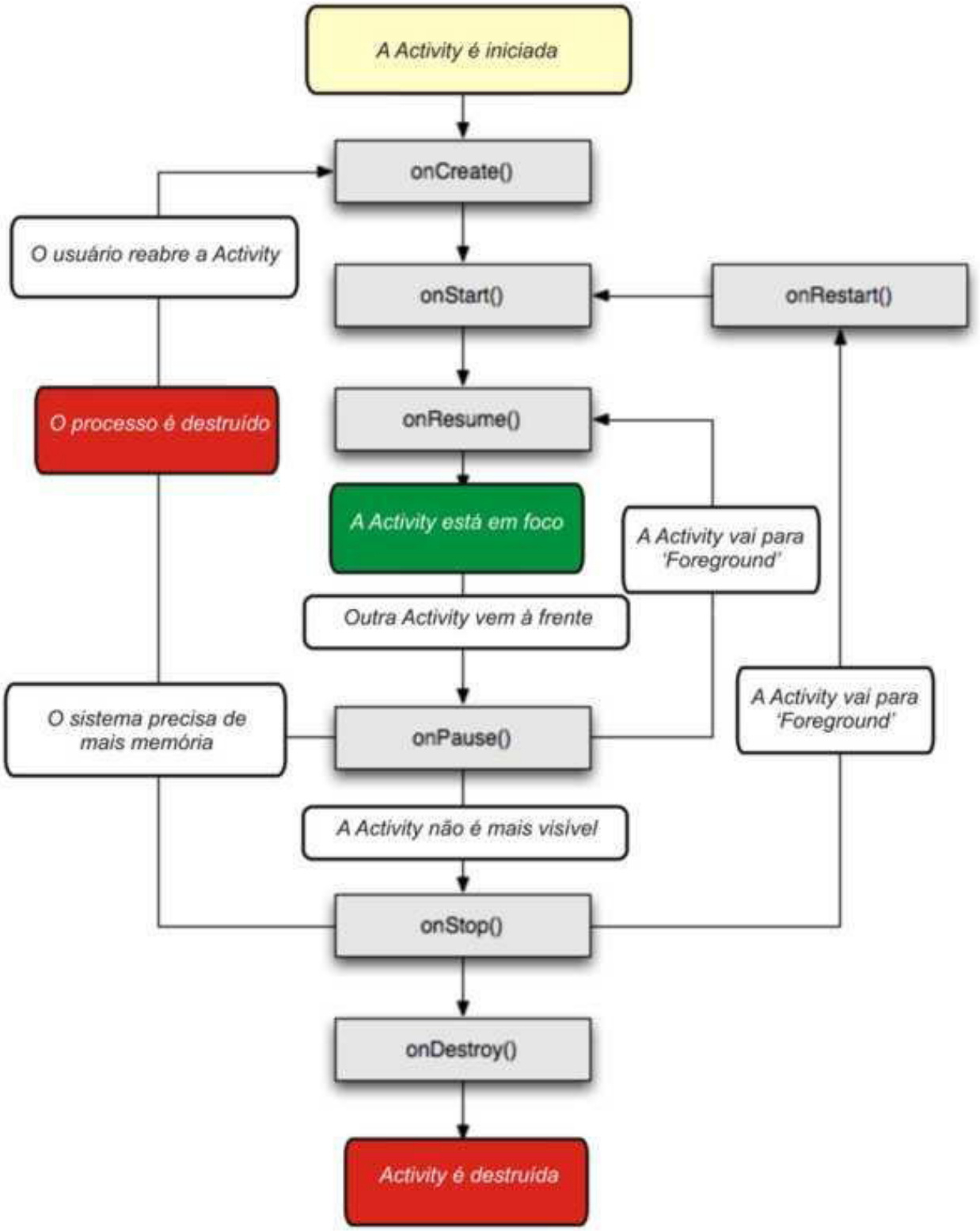


Figura 3.7 - Ciclo de vida de uma atividade (Extraído de Android Developers, 2012).

Acompanhando o fluxo do diagrama temos:

- **onCreate() –** Chamado quando uma *Activity* é criada pela primeira vez. É nela que você pode fazer a configuração estática, por exemplo, vincular dados com um *Bundle*, criar *view*, setar o layout da *Activity* etc. Após sua execução sucedida, sempre será chamado o onStart();
- **onRestart() –** Chamado sempre quando uma *Activity* é interrompida antes de ser iniciada novamente. Após sua execução sucedida, será chamado o onStart();
- **onStart() –** Chamado quando uma *Activity* está se tornando visível para o usuário. Segue para o método onResume();
- **onResume() –** Chamado quando a *Activity* começa a interagir com o usuário. Sua *Activity* está no topo da pilha de *Activities*, com a entrada do usuário apontando para ele, ou seja, é a *Activity* que está controlando a tela que o usuário está visualizando naquele momento;
- **onPause() –** Chamado quando o sistema está prestes a retomar uma activity anterior. É normalmente utilizado pra confirmar alterações não salvar, por exemplo, persistência de dados, animações etc. Uma observação: a implementação deste método deve ser rápida, porque a próxima *Activity* não será retomada até que esse método retorne, ou finalize. Desse ponto ele vai para onResume() se a *Activity* vai para frente, ou seja, é mostrada na tela para o usuário ou vai para onStop() caso o processo necessite ser destruído, por falta de memória por exemplo;
- **onStop() –** Chamado quando a *Activity* já não é visível para o usuário, porque outra *Activity* foi retomada e está cobrindo essa Activity. Isto pode acontecer se, uma nova *Activity* está sendo criada, se a *Activity* já criada está trazendo outra *Activity* à frente desta ou se a *Activity* está sendo destruída. Vai para onRestart() se a *Activity* está voltando para interagir com o usuário, ou para onCreate() caso seja criado uma nova *Activity* ou para onDestroy() se essa *Activity* está sendo fechada;
- **onDestroy() –** Destrói a *Activity* para liberar espaço em memória.

Durante a execução dos métodos onStop(), onPause(), onDestroy() o processo de uma *Activity* pode ser destruído pelo sistema operacional, caso as condições de memórias estejam em processo crítico.

3.4.4 O arquivo AndroidManifest.xml

O arquivo AndroidManifest.xml é o principal arquivo do projeto, onde ficam todas as configurações do mesmo. Ele obrigatoriamente deve ficar na pasta raiz do projeto, contendo todas configurações necessárias para executar a aplicação, como o nome do pacote utilizado, o nome das classes de cada *Activity* e várias outras configurações.

É nele por exemplo que definimos a versão do projeto, versão do SDK utilizado no projeto, a versão mínima do Android necessário para executar o projeto; adiciona também as regras de permissões para acesso a determinado hardware do celular, como o Wi-Fi, câmera, *bluetooth*, etc.

A Figura 3.8 exemplifica um arquivo AndroidManifest que possui uma *Activity* principal chamada de *"Main"* e que permite o acesso à Internet e ao estado do Wi-Fi, ou seja, a aplicação pode requerer o estado do Wi-Fi para saber se o mesmo está ligado ou desligado.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.exampl
    android:versionCode="1" android:versionName="1.0">

    <uses-sdk android:minSdkVersion="14" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <!-- Market filtering -->
    <uses-feature android:name="android.hardware.wifi.direct" android:required="true"/>

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@android:style/Theme.Holo">
        <activity
            android:name=".WiFiDirectActivity"
            android:label="@string/app_name" android:launchMode="singleTask">
            <intent-filter>
                <action
                    android:name="android.intent.action.MAIN" />
                <category
                    android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <!-- Used for transferring files  after a successful connection -->
        <service android:enabled="true" android:name=".FileTransferService" />

    </application>
</manifest>
```

Figura 3.8 - O Arquivo AndroidManifest.xml.

3.4.5 Interface com o usuário

As interfaces do usuário são construídas nos arquivos *xml*, que ficam dentro da pasta layout do projeto, usando objetos *View* e *ViewGroup*. Esses objetos são definidos hierarquicamente em forma de uma árvore na plataforma Android, como pode ser visto na Figura 3.9.

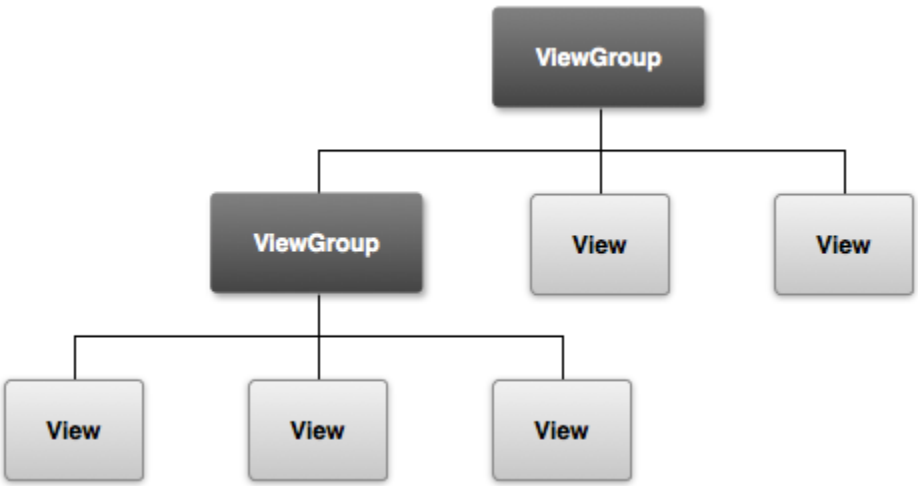


Figura 3.9 - Hierarquia de uma View e Viewgroup (Android Developers, 2012).

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 como botões, barras de status, caixa de texto, 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 (AROUCHA, 2012).

Uma forma muito comum para definir o *layout* e expressar sua hierarquia é editando diretamente o arquivo XML referente àquele *layout.* Com isso o programador ganha mais flexibilidade e controle sobre os componentes que deseja colocar na tela do aplicativo.

A figura 3.10 demonstra um arquivo *xml* de layout definindo um layout linear e três *widgets*, um *TextView*, um *EditText* e um *Button*.

```
<?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 3.10 - Exemplo de arquivo XML de layout (SOUSA, 2011, p. 24).

A Figura 3.11 mostra como essa interface é exibida na tela de um emulador:

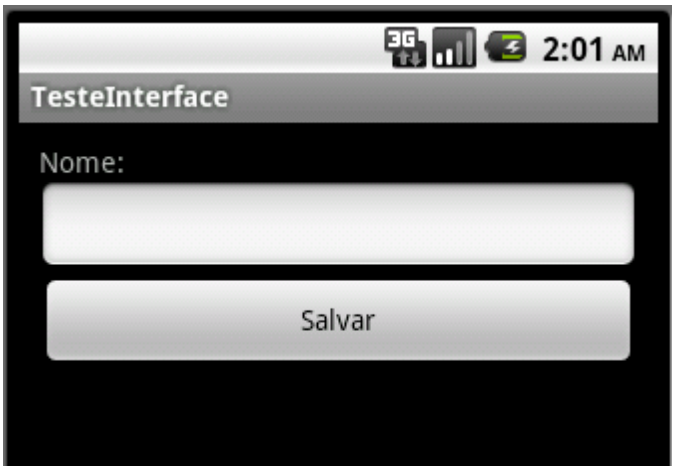


Figura 3.11 - Exemplo de interface desenvolvida utilizando XML (SOUSA, 2011, pag. 24).

3.5 Considerações Finais

Em seu trabalho, AROUCHA (2012) levantou um ponto que mostrava que uma pesquisava realizada pela empresa GARTNER INC (2012) afirmava que até o final de 2012 os dispositivos baseados na plataforma Android iriam comandar 49 por cento de todo o mercado de venda de *smartphones,* dada a grande diversidade de dispositivos móveis para esta plataforma, além do seu custo ser menor e ter um amplo suporte oferecido pela Google. No final do mesmo ano, foi constatado que os resultados foram acima do esperado, superando a porcentagem estimada e atingindo 72 por cento de todo o mercado de venda.

Em 2013, pesquisas realizadas pela empresa GARTNER INC (2013) mostraram que a plataforma Android continuou a aumentar sua liderança, conquistando 79 por cento do mercado no segundo trimestre.

Um rico ambiente de desenvolvimento somado a uma grande quantidade de dispositivos de custo baixo motiva desenvolvedores e usuários finais a cada vez mais procurarem o Android como plataforma no seu dia a dia. Esses fatores mostram que a previsão para o futuro da computação é a mobilidade, futuro esse que hoje já é realidade, e que investir hoje em plataformas móveis torna-se uma etapa essencial na formação dos desenvolvedores.

4 SEGURANÇA DE REDE

No campo de redes de computadores, a área de segurança de rede consiste na provisão e políticas adotadas pelo administrador de rede para prevenir e monitorar o acesso não autorizado, uso incorreto, modificação ou negação da rede de computadores e dos seus recursos associados. Segurança de rede envolve a autorização de acesso aos dados de uma rede, os quais são controlados pelo administrador de rede. Usuários escolhem ou são lhes atribuídas uma identificação e uma senha, ou outra informação de autenticação que permite que eles acessem as informações e programas dentro de sua autorização. A segurança de rede cobre uma variedade de redes de computadores, tanto públicas quanto privadas, que são utilizadas diariamente conduzindo transações e comunicações entre empresas, agências governamentais e indivíduos. Redes podem ser privadas, como as de uma companhia ou podem ser abertas para acesso público. Segurança de rede está envolvida em organizações, empresas e outros tipos de instituições.

A segurança de rede engloba diversas áreas, onde as principais são:

- Criptografia de Chaves e Dados;
- Vulnerabilidade em Sistemas Distribuídos;
- Vulnerabilidade em Redes Locais e de Grande Escala;
- Firewalls;
- Sistemas de Detecção de Intrusos;
- Redes Privadas Virtuais (VPN);
- Segurança em Redes sem fios;
- Controle de roteamento.

4.1 Criptografia

A criptografia é um conjunto de técnicas para esconder informação de acesso não autorizado (TECHTUDO, 2012). O objetivo principal é transformar um conjunto de informação e dados legíveis, como um arquivo de texto ou e-mail, por exemplo, em um conjunto embaralhado de caracteres impossíveis de serem compreendidos. O conceito de chave é que apenas quem possui a chave de decriptação seja capaz de recuperar os dados criptografados de forma que fiquem legíveis (TECHTUDO, 2012). Um invasor, mesmo conhecendo todo o processo de encriptação e decriptação, não consegue descobrir a informação que está sendo transmitida sem a chave de decriptação.

Um dos métodos mais antigos de criptografia que se tem notícia é o Código de César (SINGH, 2008, p.26). Seu funcionamento (Figura 4.1) era basicamente o deslocamento das letras do alfabeto de acordo com a chave. Se a chave era 3, transformava-se a letra B em E, a letra A em D, a letra C em F, e assim por diante, como mostra a figura abaixo. Porém esse código é extremamente inseguro, pois existem apenas 26 variações possíveis, dado que o alfabeto tem 26 letras.

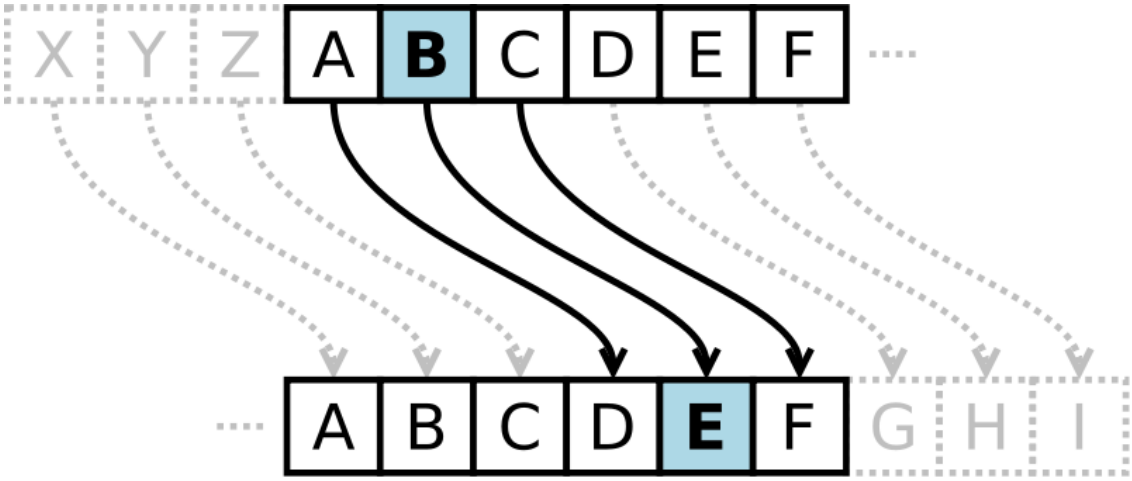


Figura 4.1 - Exemplo de funcionamento da técnica de Código de César (WIKIPÉDIA, 2013).

4.1.1 Criptografia Simétrica

Da antiguidade até os dias de hoje, muito se evoluiu. Porém, até a década de 1970, um conceito se manteve firme entre as diversas técnicas de criptografia. A mesma chave era usada para esconder e para revelar os dados. Isso é o que é chamado de criptografia simétrica (STALLINGS, 2008).

Com a evolução desse paradigma ao longo dos séculos, hoje temos mecanismos seguros e eficientes, como o 3DES (Triple Data Encryption Standard) e o AES (Advanced Encryption Standard).

Atualmente, os dois protocolos mais usados para proteção de dados na Internet, o SSL (Secure Sockets Layer) e o TLS (Transport Layer Security) utilizam a criptografia simétrica para proteger os dados transmitidos e armazenados (TECHTUDO, 2012). Porém a criptografia simétrica possui um desafio importante e impossível de ser resolvido. Como combinar uma chave secreta entre duas pessoas que querem se comunicar através da Internet? Essa pergunta não teve solução até a década de 1970 e não foi na criptografia simétrica que a solução foi encontrada, como veremos na Seção 4.2.2. A Figura 4.2 demonstra o funcionamento da criptografia simétrica.

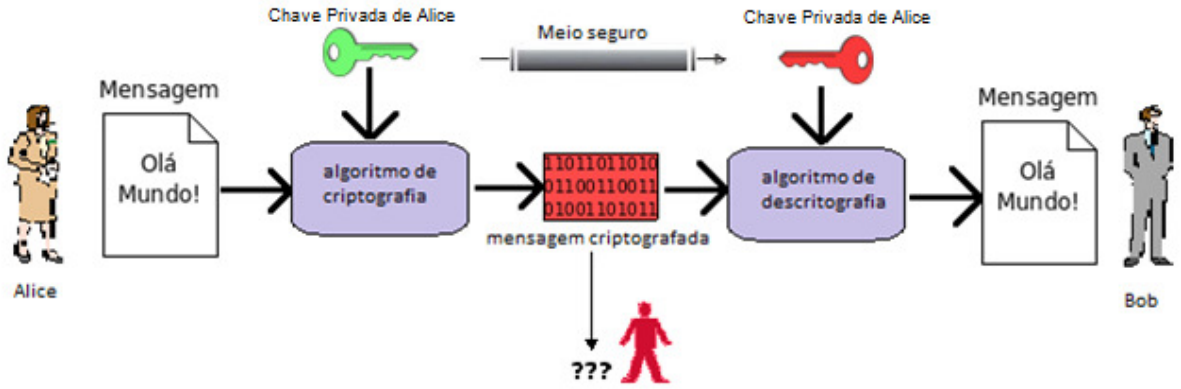


Figura 4.2 - Exemplo de funcionamento da criptografia simétrica.

Alice possui uma mensagem de e-mail que precisa enviar de forma segura para Bob. Usando a criptografia simétrica, No lado emissor, Alice gera uma chave pública e efetua a criptografia do seu e-mail junto com essa chave. Nesse momento a chave é transmitia a bob por algum meio seguro e a mensagem de e-mail criptografada é transmitida normalmente pela rede. No lado receptor, Bob efetua a descriptografia da mensagem usando a chave pública de Alice, a qual ele recebeu por algum meio seguro definido pelos dois. Isso garante que qualquer intruso que interceptar a comunicação entre Alice e Bob não consiga identificar o conteúdo da mensagem que está sendo transmitida, pois o mesmo não possui a chave pública de Alice.

Desenvolvido em 1977, um dos principais algoritmos de criptografia simétrica é o DES (*Data* Encryption Standard), 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 (AROUCHA, 2012).

4.1.2 Criptografia Assimétrica

Na criptografia simétrica (Seção 4.2.1) levantamos um desafio importante que até aquele momento era dado como impossível de ser resolvido: como combinar uma chave secreta entre duas pessoas que querem se comunicar através da Internet.

A solução para esse desafio foi dada pela criptografia assimétrica, na qual utiliza-se duas chaves distintas, mas que se complementam. Por essa característica que dá-se o nome de par de chaves, que é composto pela chave pública e pela chave privada. A chave pública é liberada para todos que desejam se comunicar com o emissor da chave enquanto a chave privada fica em poder de quem a emitiu (TECHTUDO, 2012).

A Figura 4.4 demonstra o funcionamento da criptografia assimétrica.

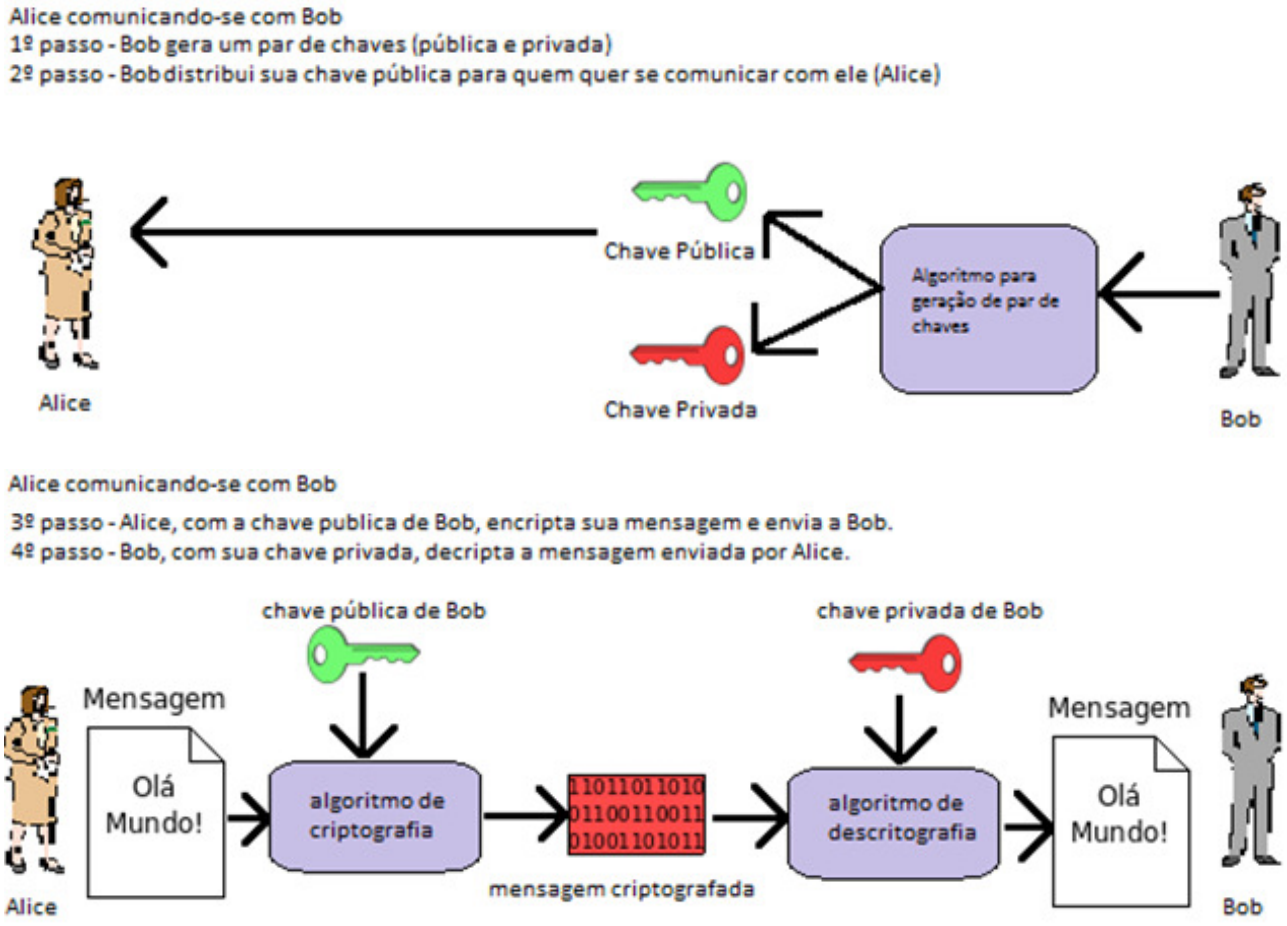


Figura 4.3 - Exemplo de funcionamento da criptografia assimétrica.

Nesse esquema, Alice quer enviar um e-mail, cifrado com criptografia assimétrica, para Bob. Para que isso ocorra, primeiramente Bob deve gerar seu par de chaves (pública e privada) e enviar sua chave pública para Alice. Após isso Alice, de posse da chave

pública de Bob e um algoritmo para criptografia assimétrica, cifra a mensagem usando essa chave pública, e envia a mensagem cifrada a Bob. Por sua vez, Bob de posse de sua chave privada (que somente ele possui e nunca circula na rede) efetua a decriptação da mensagem normalmente.

Um dos algoritmos de chave pública mais popular é o RSA que deve seu nome a três professores do Instituto MIT, Ronald **R**ivest, Adi **S**hamir e Leonard **A**dleman, que inventaram este algoritmo. É considerado um dos mais seguros, já que mandou por terra todas as tentativas de quebrá-lo. Foi também o primeiro algoritmo a possibilitar criptografia e assinatura digitais, e uma das grandes inovações em criptografia de chave pública. (DURANTE, 2004). Esse algoritmo funciona da seguinte maneira:

1. Escolha de forma aleatória dois números primos grandes p e q, de ordem 10^{100} no mínimo.
2. Compute $n = p*q$.
3. Compute $z = (p - 1)*(q - 1)$
4. Escolha um número inteiro e menor que n e que não possua fator comum com z além de1, para utilizá-lo na criptografia.
5. Para descriptografia, compute d tal que $e*d - 1$ seja divisível por z.

Por fim, temos:

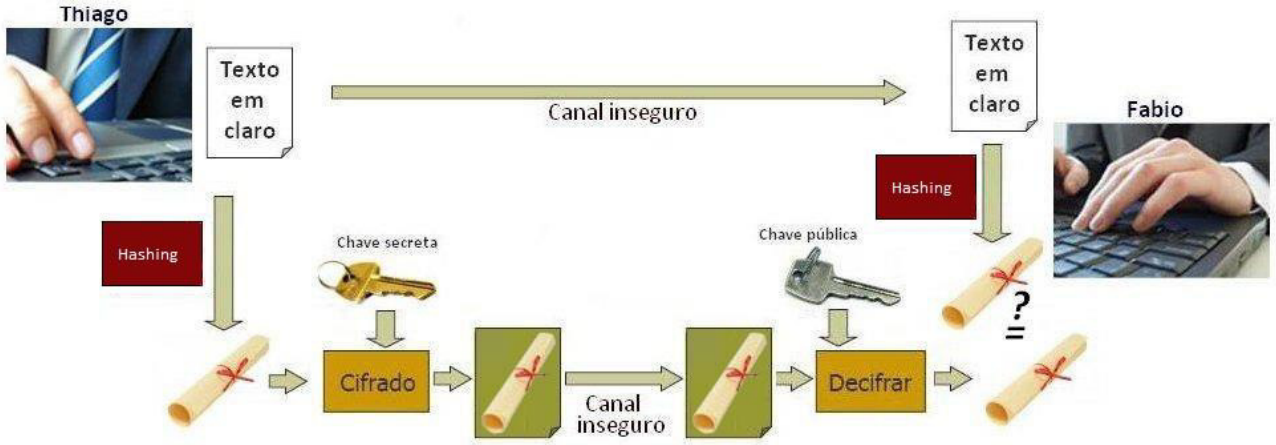
- A chave pública: o par de números (n, e).
- A chave privada: o par de números (n, d).

4.1.3 Assinatura Digital

A criptografia assimétrica não substitui a criptografia simétrica, pois eles são lentos e vulneráveis a alguns ataques. Geralmente a criptografia assimétrica é usada para distribuir com segurança as chaves simétricas, pois esta será usada para cifrar as mensagens. Essa técnica é ilustrada na Figura 4.4 e funciona da seguinte forma:

1. Thiago escreve uma mensagem a Fabio e logo lhe aplica uma função hash e cifra o resumo obtido com a sua chave privada. Antes de enviar a mensagem a Fabio, Thiago adiciona este resumo cifrado à sua mensagem.

2. Quando Fabio recebe a mensagem, primeiramente volta a gerar um resumo da mensagem que chegou e depois decifra, com a chave pública de Thiago, a mensagem com o resumo que o mesmo lhe enviou.
3. Por último compara as duas funções resumo. Se forem iguais, assegura-se de que a mensagem não foi modificada e que foi Thiago que enviou.



4. Figura 4.4 - Exemplo de assinatura digital assimétrica utilizando funções *hash*.

Todo esse processo é chamado de **Assinatura Digital** e é a forma de provar se determinada mensagem é de determinada pessoa e não de um terceiro passando-se por ela. Para verificar este requisito, uma assinatura digital deve ter as seguintes propriedades (TRINTA e MACÊDO, 2013):

- Autenticidade – o receptor deve poder confirmar que a assinatura foi feita pelo emissor.
- Integridade – qualquer alteração da mensagem faz com que a assinatura não corresponda mais ao documento.
- Irretratabilidade – o emissor não pode negar a autenticidade da mensagem.

4.1.4 Funções de Hash

Criptografar e descriptografar mensagens são processos muito dispendiosos, utilizar um resumo da mensagem original para ser criptografado é mais viável quando tudo que se quer detectar é se a mensagem original foi modificada. Existem vários algoritmos para calcular o resumo de uma mensagem, conhecidos como funções de Hash (H), o que devem

possuir 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. Para verificar se a mensagem m recebida pelo destinatário não fora modificada por um intruso ou por acidente, é utilizada a função *Hash* na mensagem e comparado o conteúdo da mensagem descriptografada com a chave pública do emissor, caso sejam iguais a mensagem enviada é integra e não repudiável (AROUCHA, 2012).

Os algoritmos de H*ash* mais utilizados são os de 16 bytes MD4 e MD5 ou o SHA-1, de 20 bytes. Segue as características de alguns algoritmos:

1. MD4: Desenvolvido em 1990/01 por Ron Rivest, vários ataques foram detectados, o que fez com que o algoritmo fosse considerado frágil.
2. MD5: O MD5 (Message-Digest algorithm 5) é um algoritmo de hash de 128 bits unidirecional desenvolvido pela RSA Data Security, Inc., muito utilizado por softwares com protocolo par-a-par (P2P), verificação de integridade e logins. Já existem alguns métodos de ataque divulgados para o MD5.
3. SHA-1 (*Secure Hash Algorithm*): Desenvolvido pelo NIST e NSA. Já foram exploradas falhas no SHA.
4. WHIRLPOOL: função criptográfica de *Hash* desenvolvida por Paulo S. L. M. Barreto e por Vincent Rijmen (co-autor do AES). A função foi recomendada pelo projeto NESSIE (Europeu). Foi também adotado pelo ISSO e IEC como parte do padrão internacional ISSO 10118-3.

4.2 Sockets Seguros (Secure Sockets Layer – SSL)

O *Secure Socket Layer* é um protocolo de comunicação que implementa um canal seguro para a comunicação entre aplicações na Internet, de forma transparente independentemente da plataforma.

Sua primeira versão foi desenvolvida pela Netscape Communications em julho de 1994. A referência para a implementação da versão 2 foi lançada em abril de 1995, sendo distribuída junto com os browsers Netscape e Internet Explorer, e os servidores web mais

comuns – Apache, IIS, Netscape Server, NCSA etc. Transformou-se em um padrão para *e-commerce,* tendo sua especificação submetida ao grupo de trabalho W3C.

A versão 3 do SSL foi lançada em novembro de 1996, tendo como melhorias a escolha de cifras e compressão por parte do servidor, suporte mais completo para a troca de chaves de algoritmos de cifragem, possibilidade de renegociação das cifras em uso e a separação das chaves de autenticação e encriptação. Serviu de base para o desenvolvimento do TLS versão 1.0, um protocolo padronizado da IETF (*Internet Engineering Task Force*) originalmente definido pelo RFC 2246. Grandes instituições financeiras como Visa, MasterCard, American Express, dentre outras, aprovaram o SSL para comércio eletrônico seguro na Internet (COMODOBR, 2013).

Segundo (STALLINGS, 2008), o SSL possui dois conceitos fundamentais:

- Sessão SSL: é a associação entre o cliente e o 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.

4.2.1 Arquitetura do protocolo SSL

O SSL atua entre as camadas de transporte (TCP) e de aplicação, sendo independente do protocolo de alto nível podendo funcionar sobre os protocolos HTTP, Telnet, FTP, SMTP entre outros, de forma transparente.

Ele implementa duas novas camadas, sobre o TCP/IP, conforme mostra a Figura 4.5.

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 4.5 - Camadas implementadas pelo SSL.

SSL Change Cipher SPEC Protocol: Sinaliza as transições nas estratégias de cifragem. Constitui-se de uma única mensagem que pode ser transmitida tanto pelo cliente quanto pelo servidor para notificar que os próximos blocos utilizam chaves de encriptação recém negociadas.

SSL Alert Protocol: Acompanha os erros na Record Layer, fazendo troca de mensagens para sinalizar problemas com a sequência de mensagens, erros de certificação ou encriptação.

SSL Handshake Protocol: faz a autenticação entre cliente e servidor, cuidando da inicialização da comunicação, permitindo a negociação do Código de Autenticação de Mensagens (MAC), algoritmo de encriptação e as chaves criptográficas iniciais. Utiliza as chaves assimétricas para fazer a negociação inicial, abrindo um canal seguro para o envio da chave simétrica da sessão. Todas as mensagens da negociação utilizam o MAC e funções de hash (como SHA, MD5) para aumentar a segurança do processo inicial.

SSL Record Layer Protocol: Encapsula as camadas de nível mais alto (implementa o HTTPS, quando conjugado com o HTTP), provendo serviços de fragmentação (transforma blocos de dados em registros SSLPlaintext), compressão (transforma os registros SSPlaintext em registros SSLCompressed, utilizando os algoritmos negociados no handshake), autenticação de mensagens, acréscimo do MAC e número sequencial e encriptação (as funções definidas no Handshake são definidas na mensagem SSLChipherSpec e são utilizadas para transformar o SSLCompressed em SSLCiphertext).

4.2.2 Funcionamento do protocolo SSL

O processo para o estabelecimento da comunicação é ilustrado na Figura 4.6 e descrito a seguir:

1. O cliente envia uma mensagem “olá” ao servidor que lista as capacidades criptográficas do cliente, como a versão do SSL, os cipher suites e os métodos de compactação de dados suportados pelo cliente. A mensagem também contém um número aleatório de 28 bytes.
2. O servidor responde à mensagem “olá” do cliente que contém o método criptográfica (cipher suite) e o método de compactação de dados selecionado pelo servidor, o ID da sessão e outros números aleatórios.

3. O servidor envia seu certificado digital. (O servidor utiliza certificados digitais X.509 V3 com SSL).
4. O servidor envia uma mensagem de servidor “olá enviado” e aguarda uma resposta do cliente.
5. Ao receber a mensagem de servidor “olá enviado”, o cliente (navegador da Web) verifica a validade do certificado digital do servidor e verifica se os parâmetros de “olá” no servidor são aceitos.
6. O cliente envia uma mensagem “intercâmbio de chave do cliente”. Essa mensagem contém o segredo pré-master, um número aleatório de 46 bytes utilizado na geração de chaves criptografia simétrica e chaves de MAC (*códigos de autenticação de mensagens*), criptografados com a chave pública do servidor.
7. O cliente utiliza uma série de operações criptográficas para converter um segredo pré-master em um segredo master, a partir do qual todos os materiais de chave exigidos para criptografia e para autenticação de mensagem é derivado. Então o cliente envia uma mensagem “alterar cipher spec” para fazer o servidor ir para o chipher suite negociado mais recente. A mensagem de texto enviada pelo cliente (a mensagem “concluído”) é a primeira mensagem criptografada com esse método cipher e essas chaves.
8. O servidor responde a mensagem “alterar cipher spec” e “concluído” por si só.
9. O protocolo de reconhecimento SSL é concluído e os dados do aplicativo criptografados podem ser enviado.

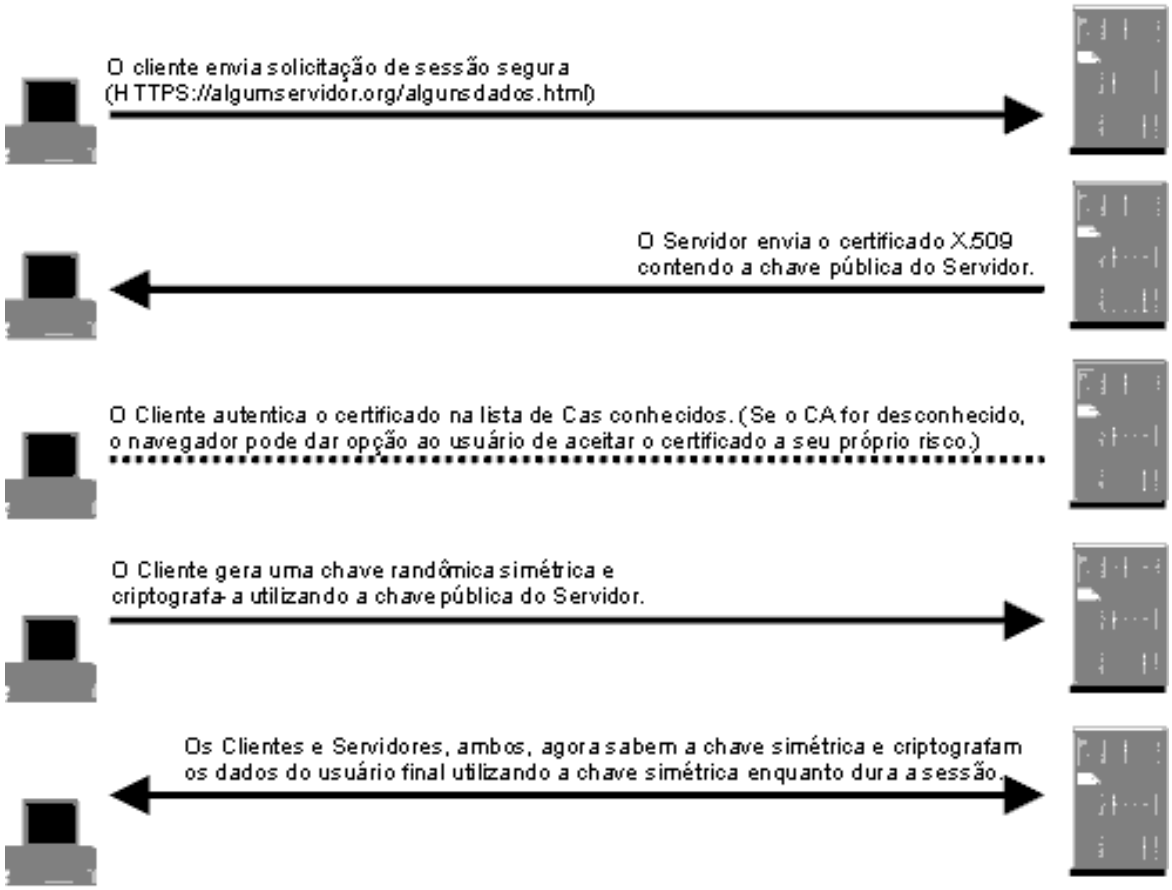


Figura 4.6 - Protocolo de Reconhecimento SSL com Autenticação de Servidor.

4.2.3 Transporte Seguro (Transport Layer Security – TLS)

O TLS 1.0 (RFC 2246, 2013) é um protocolo de segurança padronizado para a Internet, desenvolvido pela IETF baseada na versão 3.0 do SSL (STALLINGS, 2008).

Os objetivos do protocolo TLS, em ordem de prioridade são:

- Segurança com criptografia: TLS deve ser usado para estabelecer uma conexão segura entre duas partes.
- Interoperabilidade: Programadores independentes deve conseguir desenvolver aplicações utilizando TLS que possam trocar parâmetros criptográficos sem um conhecer o código do outro.
- Extensibilidade: TLS busca o fornecimento de uma estrutura (framework), em que novos métodos de criptografia simétrica e assimétrica podem ser adicionados, sem a necessidade de implementação de uma nova biblioteca de segurança.
- Eficiência Relativa: Operações de criptografia, principalmente de chave pública, exigem um alto processamento. Sendo assim, o protocolo TLS incorporou um

mecanismo de armazenamento para evitar que toda conexão ao ser estabelecida não precise processar operações de criptografia. Com isso, reduz-se também a atividade da rede.

As principais 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 (AROUCHA, 2012).
- 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 utiliza apenas o MAC.
- Algoritmos para troca de chaves: o TLS suporta todos os algoritmos da SSL 3.0, exceto o Fortezza (AROUCHA, 2012).

4.3 JSSE

O *Java Secure Socket Extension* (JSSE) é um conjunto de APIs, ferramentas e implementações de algoritmos de segurança usados frequentemente, mecanismos e protocolos. As APIs de segurança do Java abrangem uma ampla gama de áreas, incluindo criptografia, infraestrutura de chave pública, comunicação segura, autenticação e controle de acesso. A tecnologia de segurança do Java fornece ao desenvolvedor uma estrutura de segurança completa para a escrita de aplicações e também fornece ao usuário ou administrador um conjunto de ferramentas para gerenciar com segurança as aplicações.

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

a) Implementado 100% em Java
b) Pode ser exportado para a maioria dos países
c) Fornece suporte para API SSL versão 2.0 e 3.0 e a implementação para a versão 3.0 do SSL
d) Provê suporte a API TLS e uma implementação da versão 1.0
e) Possui classes que podem ser instanciadas para criar canais seguros (SSLSocket, SSLServerSocket e SSLEngine)
f) Fornece suporte a autenticação do cliente e servidor, que é parte do *handshaking* normal da SSL
g) Oferece suporte a negociação *cipher suíte*

h) Suporte a vários algoritmos de criptografia usados em conjunto às *cipher suites*

A Tabela 5.2 descreve cada característica de segurança em mais detalhes e aponta para seus recursos com mais informações.

Tabela 5.1 - Recursos disponibilizados pelo JSSE.

Recursos de alto nível	Recursos de baixo nível	Benefícios
Segurança de Plataforma	Recursos de segurança embutidos na linguagem imposta pelo compilador Java e a máquina virtual • Dados de tipagem forte • Gerenciamento automático de memória • Verificação de bytecode • Classe de carregamento segura	Fornece uma plataforma confiável e segura para o desenvolvimento e execução de aplicativos. Dados em tempo de compilação, verificação de tipo, gerenciamento automático de memória leva a um código mais robusto e reduz a corrupção de memória e vulnerabilidades. Verificação de bytecode garante o código estar em conformidade com a especificação JVM e impede que um código hostil corrompa o ambiente de execução. Carregadores de classe garantem que o código não confiável não pode interferir o funcionamento de outros programas Java.
Criptografia (recurso usado neste trabalho)	• API abrangente, com suporte para uma ampla gama de serviços de criptografia, incluindo assinaturas digitais, *message digests,* cifras (simétricas, assimétricas, *stream & block*), códigos de autenticação de mensagens, geradores de chaves e fábricas de chaves • Suporte para uma ampla variedade de algoritmos padrão, incluindo RSA, DAS, AES *Triple* DES, SHA, PKCS#5, e RC4 • Suporte ao *token* criptográfico PKCS#11	Fornece uma API extensível, cheia de recursos para a construção de aplicações seguras. • Algoritmos e implementação independente • Arquitetura baseada em Provedor
Autenticação e Controle de Acesso	• APIs de autenticação abstrata que pode incorporar uma ampla gama de mecanismos de *login* através de uma arquitetura	Permite simples autenticações de vários mecanismo e acesso a recursos com base na identidade do usuário ou assinante do código. Recente suporte (no JDK 5) para

	plugável • Uma política agrangente e API de permissões que permite ao desenvolvedor criar e administrar aplicações que requerem acesso detalhado aos recursos sensívei à segurança	assinaturas protocoladas torna mais fácil a implantação de código assinado, evitando a necessidade de reassinar o código quando o certificado expirar
Comunicações seguras	APIs e implementações para as seguintes normas baseadas em protocolos de comunicações seguras: *Transport Layer Security* (TLS), *Secure Sockets Layer* (SSL), *Kerberos* (acessível atrave´s de GSS-API) e *Simple Authentication and Security Layer* (SASL). Suporte completo para HTTPS sobre SSL/TLS	Autentica pares sobre uma rede não confiável e protege a integridade e privacidade dos dados transmitidos entre eles
Infraestrutura de chaves Públicas, *Public Key Infrastructure* (PKI)	Ferramentas para gerenciamento de chaves e certificados, e abstratas APIs abrangentes com suporte para os seguintes recursos e algoritmos: • *Certificates and Certificate Revocation Lists* (CRLS)*:* X.509 • *Certification Path Validators and Builders:* PKIX (RFC 3280), *On-line Certificate Status Protocol* (OCSP) • *KeyStores:* PKCS#11, PKCS#12 • *Certificate Stores* (Repositórios): LDAP*, java.util.Collection*	Facilita o desenvolvimento e implantação de complexas aplicações PKI. Recente suporte (no JDK 5) para OCSP fornece um método mais escalável e oportuna aplicações verificar o estado da revogação do certficado

4.4 Considerações Finais

Desde o surgimento da Internet, a segurança de dados é um dos principais pontos a ser analisado, pois diariamente o volume de troca de dados é imenso, e existem muitos usuários que utilizam de conhecimento e falhas de segurança para se apoderar desses dados, tendo a possibilidade de usá-los como quiser. Grandes empresas de tecnologias como Google, Apple e Microsoft investem pesado em segurança digital anualmente.

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 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).

5 REDE P2P DROID SEGURA

Nesse capítulo é mostrada a implementação de uma rede P2P para Android, denominada P2P Droid e também as funcionalidades de um cliente executando sobre essa rede, chamado de Cliente P2P Droid.

A rede P2P Droid é uma arquitetura de rede P2P que provê aos usuários um meio de comunicação e compartilhamento de recursos. Foi especificada e implementada sobre a API Wi-Fi P2P Manager (Android Developers, 2013) criada pela Google.

O Cliente P2P Droid é uma aplicação Android que executa na versão 4.0 ou superior desse Sistema Operacional. Implementa as principais funcionalidades da rede P2P Droid e permite a comunicação e transferência de dados entres diferentes dispositivos que estejam funcionando sobre dessa rede.

5.1 O Padrão de Comunicação *Wi-Fi Direct*

O Sistema Operacional Android, a partir da versão 4.0 (*Ice Cream Sandwich*) conta com um sistema de compartilhamento de arquivo mais rápido e mais confiável que o Bluetooth, chamado de Wi-Fi *peer-to-peer* (P2P), que implementa a especificação Wi-Fi Direct (Wi-Fi Alliance, 2009) criada pela Wi-Fi Alliance (Wi-Fi Alliance, 2013).

O Wi-Fi Direct é uma tecnologia que permite transformar qualquer aparelho em um ponto de acesso. Uma nova especificação criada pela Wi-Fi Alliance que possibilita criar redes *adhoc* entre dispositivos Wi-Fi com a mesma facilidade encontrada em conexões Bluetooth. É uma reformulação total do modo como as conexões *adhoc* trabalhavam anteriormente. Em vez de precisar conectar os dispositivos a um ponto de acesso central, este novo protocolo transforma qualquer aparelho que possua a tecnologia Wi-Fi em um ponto de acesso em potencial conforme a Figura 5.1.

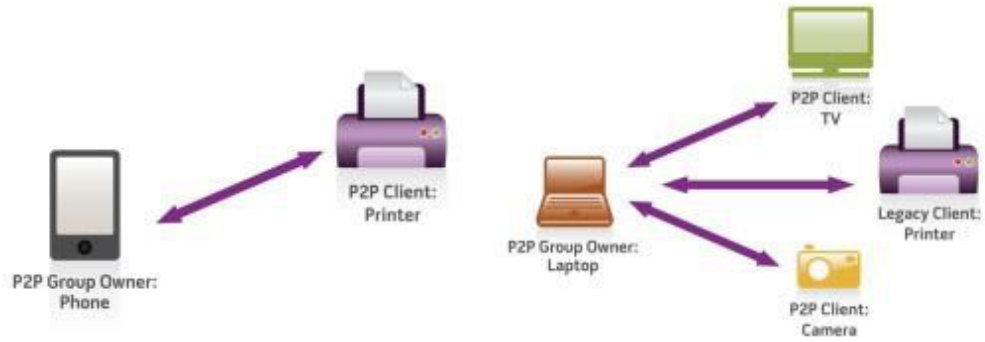


Figura 5.1 - Proposta do Wi-Fi Direct.

5.1.1 Arquitetura

Em uma típica rede Wi-Fi, os clientes procuram e se associam a redes sem fio disponíveis, que são criadas e anunciadas por Pontos de Acesso (AP). Cada um desses dispositivos tem funções que envolvem um conjunto diferente de funcionalidades. A grande novidade do Wi-Fi Direct é que esses papéis são especificados como dinâmico, e portanto, um dispositivo Wi-fi Direct pode implementar tanto o papel de um cliente quanto o papel de um AP (ou servidor). Esses papéis são, portanto, as funções lógicas que poderiam até mesmo serem executadas simultaneamente pelo mesmo dispositivo, este tipo de operação é chamado de modo simultâneo.

Os dispositivos que implementam essa tecnologia se comunicam estabelecendo Grupos P2P, os quais são funcionalmente equivalentes às redes tradicionais da infraestrutura Wi-Fi. Conforme a Figura 5.2, em uma comunicação, o dispositivo que se torna o proprietário do grupo é chamado de *P2P Group Owner* (P2P GO) e o dispositivo que atuar como cliente é conhecido como *P2P Client.*

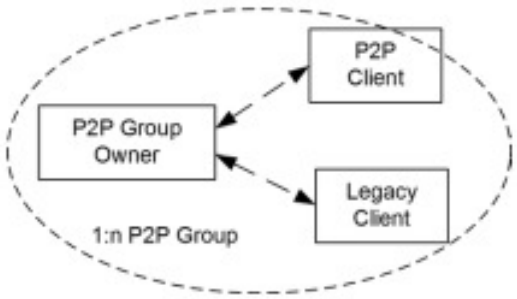


Figura 5.2 - Componentes P2P e topologia.

A divisão de papéis para saber quem será o proprietário do grupo e o cliente, funcionalmente é dinâmico, ou seja, as responsabilidades de servidor e cliente são negociadas na configuração inicial da conexão. Por exemplo, suponha dois dispositivos P2P onde cada um descobre o outro; no momento da comunicação eles negociam seus papéis (P2P cliente e P2P servidor) para estabelecer o grupo P2P. Uma vez que o grupo P2P é estabelecido, outros clientes P2P podem se juntar ao grupo como acontece em uma conexão Wi-Fi tradicional. Clientes legados também podem se comunicar com o P2P GO (*Group Owner*), contanto que eles suportem os mecanismos de segurança requeridos. Por padrão o Wi-Fi Direct usa WPA2PSK (HSC, 2010) como padrão de segurança; desta forma, os dispositivos legados não pertencem formalmente ao grupo P2P e não suportam as funcionalidades melhoradas

definidas no Wi-Fi Direct, mas eles simplesmente “enxergam” o P2P GO (*Group Owner*) como um AP (*Access Point*) tradicional.

A natureza lógica dos papéis P2P suporta diferentes implementações de arquitetura; uma delas é mostrada na Figura 5.3, onde é representado um cenário com dois grupos P2P. O primeiro cenário é um telefone móvel compartilhando sua conexão 3G com dois laptops; neste primeiro cenário, os três dispositivos formam um grupo, o telefone está agindo como o P2P GO (*Group Owner*), enquanto os dois *laptops* se comportam como clientes P2P. A fim de ampliar a rede, um dos laptops estabelece um segundo grupo P2P com uma impressora; para este segundo grupo, o laptop funciona como o P2P GO (*Group Owner*). Para atuar como cliente P2P e como P2P GO, o laptop vai tipicamente alternar entre os dois papéis pelo tempo de compartilhamento da interface Wi-FI.

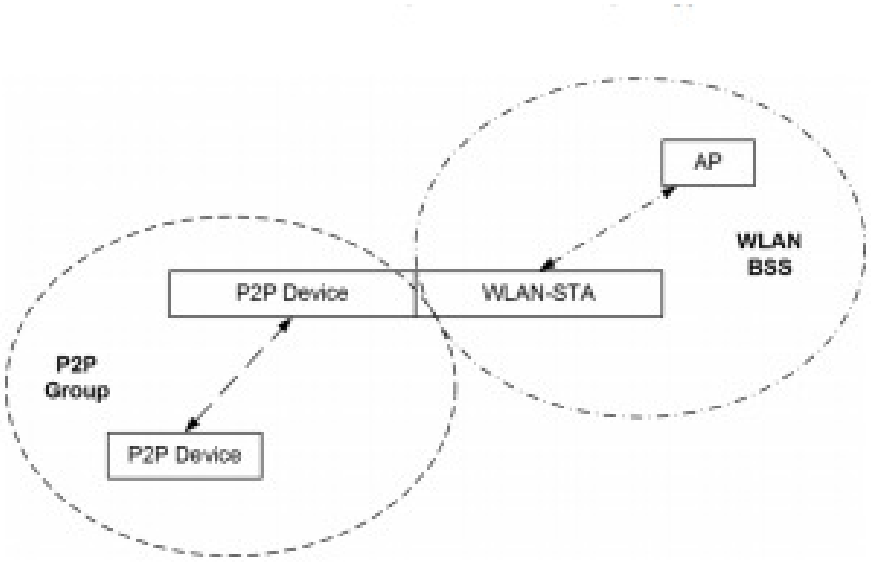


Figura 5.3 - Topologias suportadas pelo Wi-Fi Direct.

5.2 Wi-Fi P2P

O Wi-Fi P2P (Android Developers, 2013) é uma API, implementada sob a especificação Wi-Fi Direct, que permite dispositivos com Android 4.0 (API level 14) ou superior, com hardware apropriado, conectar diretamente com qualquer outro dispositivo via Wi-Fi sem a necessidade de um ponto de acesso entre eles. Usando essa API, o desenvolvedor pode descobrir e conectar a outros dispositivos que suportem o Wi-Fi Direct e então, comunica-se com uma velocidade superior e com distâncias mais longas do que uma conexão

Bluetooth. Isso é útil para aplicações que compartilham dados entre usuários, tais como jogos *multiplayer* ou uma aplicação de compartilhamento de fotos.

A API do Wi-Fi P2P é constituída das seguintes partes principais:

- Métodos que permitem descobrir, solicitar e conectar a outros pares são definidos na classe **WifiP2pManager**.
- ***Listeners*** (ouvintes) permitem que você seja notificado sobre o sucesso ou falha dos métodos que são chamados da classe **WifiP2pManager.** Quando os métodos da classe **WifiP2pManager** são chamados, cada método pode receber um **listener** específico, passado como parâmetro.
- ***Intents*** notificam você sobre eventos específicos detectados pelo **Wi-Fi Direct Framework,** como uma queda de conexão ou um novo par descoberto.

Constantemente o desenvolvedor usará esses três componentes principais da API juntos. Por exemplo, você pode fazer com que o *WifiP2pManager.ActionListener* chame o método *discoverPeers(),* então você pode ser notificado com os métodos *ActionListener.onSuccess()* e *ActionListener.onFailure()*. Na Tabela 5.1 são mostrados os principais métodos que a API Wi-Fi P2P disponibiliza ao desenvolvedor.

Tabela 5.2 - Métodos do Wi-Fi P2P.

Método	Descrição
initialize()	Registra a aplicação com o Wi-Fi Framework**.** Esse método deve ser chamado antes que seja chamado qualquer outro método do Wi-Fi Direct.
connect()	Inicia uma conexão par-a-par com outro dispositivo ao alcance.
cancelConnect()	Cancela qualquer negociação de um grupo par-a-par.
requestConnectInfo()	Solicita informações de conexão de um dispositivo.
createGroup()	Cria um grupo par-a-par com o atual dispositivo como proprietário do grupo.
removeGroup()	Remove o grupo par-a-par atual.
requestGroupInfo()	Solicita informações de um grupo par-a-par
discoverPeers()	Inicia a descoberta de pares na rede.
requestPeers()	Solicita a lista atual de pares descobertos, ou seja, que estejam ao alcance para uma conexão.

5.3 Cliente P2PDroid

Segundo (SOUSA, 2011), 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.

Como definido Sousa (2011), O cliente P2PDroid roda sobre a arquitetura P2PLAWS, criada por Ramos (2009). Essa arquitetura é baseada na rede Gnutella, totalmente descentralizada, onde todos os nós da rede possuem as mesmas funcionalidades e operações.

Nesse trabalho é apresentado um novo cliente P2PDroid, executando sobre a arquitetura Wi-Fi Direct (definida Seção 5.2).

Usando a API Wi-Fi P2P, foi implementada uma arquitetura que facilita a descoberta, conexão e troca de mensagens entre dispositivos móveis. Essa arquitetura funciona de modo semelhante ao bluetooth, onde a descoberta dos pares é limitada ao raio de abrangência do hardware envolvido na tecnologia.

Em termos de hardware, o Wi-Fi Direct trabalha com o Wi-Fi do dispositivo; isso potencializa algumas características dessa arquitetura em relação a outras, como por exemplo, o raio de abrangência para descoberta de pares, a velocidade de transferência de arquivos e a estabilidade da conexão, tornando-se assim mais robusta do que seus concorrentes (Bluetooth 3.0, por exemplo).

5.3.1 Operações Básicas

O P2PDroid define algumas operações básicas, essenciais para o funcionamento da aplicação, são elas:

a) *Initialize()*: Registra a aplicação com o Wi-Fi Framework e deve ser chamado assim que a aplicação é iniciada. Esse registro é necessário para que a aplicação possa usar os recursos de Wi-Fi do dispositivo, como por exemplo, ser notificada quando o estado do Wi-Fi mudar de ligado para desligado ou ser notificada quando receber um pedido de conexão de outro dispositivo.
b) *DiscoverPeers()*: Inicia a descoberta dos pares. Essa operação é usada para iniciar a descoberta de pares ao alcance do dispositivo.
c) *RequestPeers()*: Se a operação *discoverPeers()* detectar algum dispositivo ao alcance

d) *Connect():* Inicia uma conexão par-a-par com um dispositivo. Envia um convite de conexão a outro dispositivo, esse podendo aceitar ou recusar o pedido.

e) *Disconnect()*: Encerra uma conexão com outro dispositivo.

5.3.2 Aplicação

A aplicação P2PDroid implementa as operações básicas definidas pela arquitetura de rede Wi-Fi Direct, permitindo o compartilhamento de arquivos entre os dispositivos.

A aplicação funciona de forma semelhante ao *bluetooth*. Ao iniciar a aplicação, é mostrada uma tela com algumas informações do dispositivo, como o "apelido" do dispositivo e o seu *status* (Figura 5.4).

Figura 5.4 - Tela inicial P2P Droid.

Ao clicar em buscar (lupa) é iniciado o processo de busca pelos pares vizinhos (Figura 5.5), ou seja, outros dispositivos que estejam ao alcance para a comunicação.

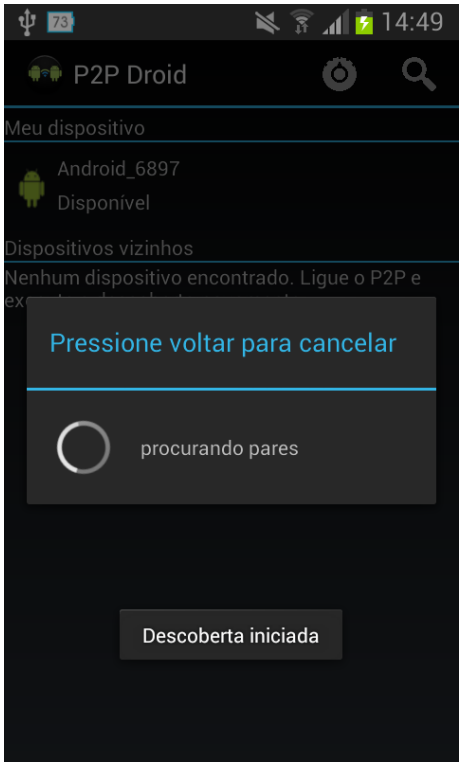


Figura 5.5 - Iniciando a descoberta por pares.

Caso sejam encontrados dispositivos para o pareamento, é mostrada uma lista com informações dos dispositivos, como o "apelido" e o *status*, como mostra a Figura 5.6.

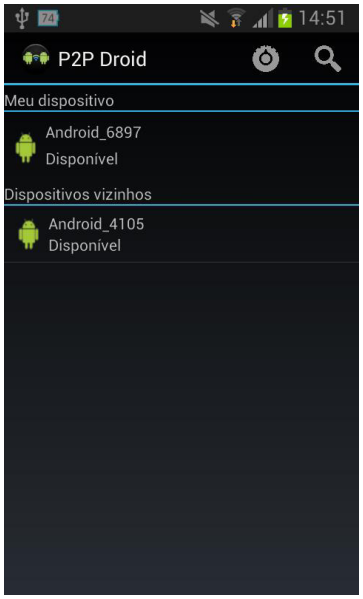


Figura 5.6 - Lista de pares que estão ao alcance.

Conforme a Figura 5.7, ao selecionar um dispositivo da lista, são mostradas algumas informações desse dispositivo vizinho, assim como o botão para iniciar a conexão.

Figura 5.7 - Informações sobre o par selecionado.

Ao clicar em conectar, o dispositivo envia um convite para o vizinho ao qual quer se conectar (Figura 5.8), e fica a critério desse vizinho aceitar ou rejeitar o convite (Figura 5.9).

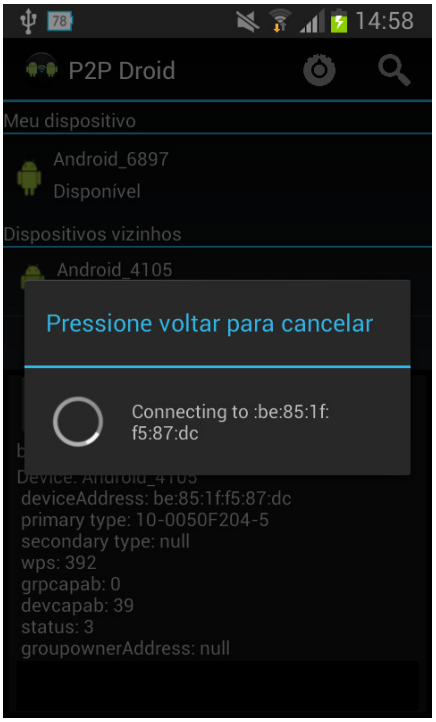


Figura 5.8 - Conectando ao par selecionado.

Figura 5.9 - Mensagem de convite de outro par.

Se o dispositivo convidado aceitar o convite, é estabelecida uma conexão p2p diretamente, sem a necessidade de um ponto de acesso entre eles. Nesse estágio, os dispositivos já podem iniciar a comunicação e transferência de arquivos, como mostra a Figura 5.10.

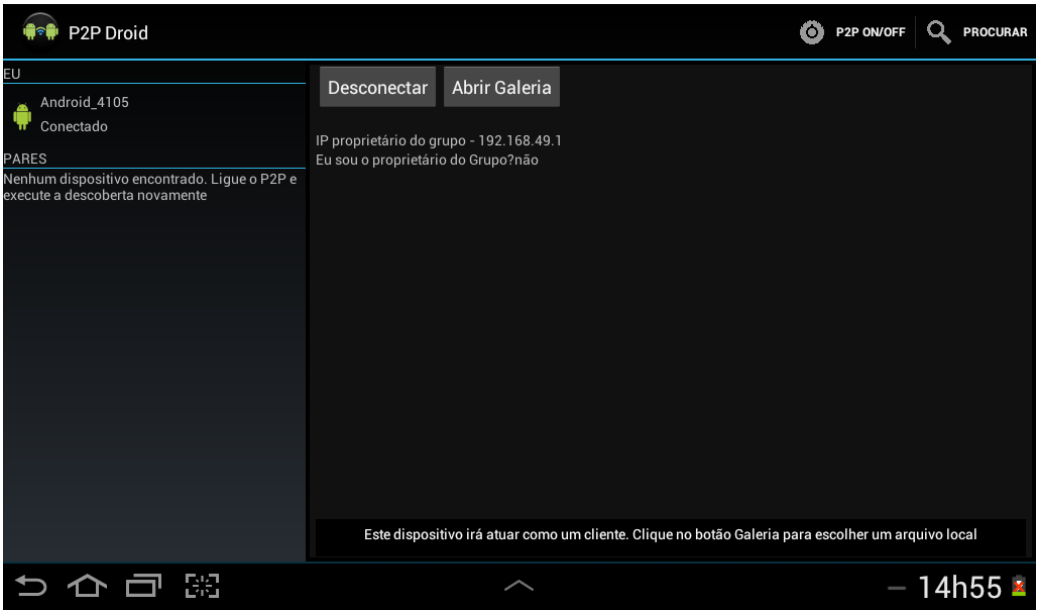


Figura 5.10 - Tela que mostra os dispositivos conectados.

5.3.3 Processo de Comunicação

Os clientes desta arquitetura realizam a comunicação através de *sockets.* A seguir é demonstrado o fluxo normal para a transferência de arquivos por meio desse tipo de comunicação:

1. O servidor cria um *ServerSocket*. Esse socket vai esperar por uma conexão do cliente em uma porta específica, que é bloqueada até que isso aconteça.
2. O cliente instancia um Socket cliente, que usa o endereço de IP e porta do servidor.
3. O cliente envia os dados para o servidor. Os dados podem ser enviados como um fluxo de bytes (*byte streams*) ou como objetos.
4. O servidor espera pela conexão do cliente. Quando a conexão acontece, o servidor recebe os dados do cliente.

É importante ressaltar aqui, que esse fluxo de transferência de arquivos é realizado sem a camada de segurança, ou seja, não é aplicado nenhum tipo de criptografia nos dados transferidos. Essa camada de segurança será discutida na Seção 5.5 onde veremos como tornar essa transferência de dados mais segura usando criptografia simétrica e assimétrica.

5.4 Camada de Segurança

Com o grande avanço da Internet e o compartilhamento de dados, principalmente arquivos, via redes P2P, houve a necessidade de se utilizar um mecanismo que garantisse a segurança desse tráfego de dados, visto que a internet proporciona um grande risco, pois as informações transmitidas podem ser facilmente interceptadas ou modificadas por terceiros, além da incerteza de saber se a pessoa com quem está se comunicando é realmente quem diz ser.

Este capítulo demonstra o desenvolvimento de uma camada de segurança para a rede P2PDroid, realizado através das APIs da JSSE (*Java Secure Socket Extension*), onde são disponibilizadas as ferramentas necessárias para os dois tipos de criptografia baseada em chaves, que serão usados neste trabalho, além de outras funcionalidades.

5.4.1 Implementação

Para a implementação da camada de segurança foi utilizado a criptografia simétrica para cifrar os dados, e a criptografia assimétrica para transferir a chave simétrica no momento da conexão, ou seja, primeiramente é transferida a chave simétrica por meio da criptografia assimétrica, para que a mesma possa ser usada na cifragem e decifragem dos dados.

Como visto no Capítulo 4 onde é discutido os tipos de criptografias, quando se deseja criptografar uma grande quantidade de dados (mais de 100 bytes), usa-se preferencialmente a criptografia simétrica (onde a mesma chave que cifra um dado serve para decifrá-lo, ou seja, a chave simétrica). Resumidamente, as vantagens da criptografia simétrica são:

a) Muito mais rápida que a criptografia assimétrica.
b) Segura para pequenas ou grandes quantidade de dados.

O algoritmo recomendado hoje em dia é o AES ou Rijndael (com chaves de tamanho 128, 192 ou 256 bits); na implementação desse trabalho é usado o AES com chave de 128 bits. Entretanto, como vimos anteriormente, para que se possa enviar um dado criptografado com criptografia simétrica, é necessário que ambas as partes (remetente e destinatário) conheçam a chave simétrica, e essa chave não pode ser enviada de qualquer forma de um para o outro a menos que uma das partes apareça pessoalmente com um envelope e dê nas mãos da outra.

Para resolver esse problema, é usado a criptografia assimétrica que tem a finalidade de criptografar a chave simétrica para que posteriormente possa ser enviada e usada na transferência dos dados. Se a chave *Pública* do destinatário é conhecida, o remetente pode mandar um dado (com menos de 100 bytes) para ele, criptografado com a chave *Pública* desse destinatário, de modo que o mesmo usa sua chave *Privada*, que ninguém conhece, para decifrá-la.

Então, por que não criptografar todos os dados que serão transmitidos com a criptografia assimétrica? É simples, dados com mais bytes não devem ser criptografados com criptografia assimétrica (por exemplo, RSA), porque isso expõe vulnerabilidades nesses algoritmos que só se manifestam quando a quantidade de dados é grande. Por se tratar de um sistema de criptografia muito utilizado, o RSA vem tendo suas vulnerabilidades pesquisadas e

analisadas praticamente desde sua publicação inicial. Uma lista bem detalhada de ataques a esse sistema pode ser encontrada no artigo de Dan Boneh[5].

Todos os recursos necessários para a implementação da camada se encontram nos pacotes *java.security* e *java.crypto*; são neles que encontramos as classes para gerar as chaves e os algoritmos de criptografia necessários para a cifragem dos dados que serão trocados na comunicação. Vale ressaltar aqui, que somente os dados são criptografados e trafegados via sockets normais, ou seja, a comunicação não é criptografada (não se usa Sockets Seguros) , pois esse tipo de comunicação necessita que os dispositivos possuam certificados digitais, os quais devem ser gerados a parte; diminuindo assim o conceito de mobilidade, pois a ideia é que em uma aplicação para usuário final, possa ser usado uma arquitetura de segurança sem a necessidade de recursos externos à aplicação, ou seja, todo o processo de criptografia deve ser dinâmico. Porém é garantido que se algum intruso interceptar algum pacote transmitido, os dados estarão totalmente criptografados.

A Figura 5.11 demonstra o esquema de comunicação entre aplicações usando a camada de segurança desenvolvida.

[5] Boneh, Dan. (1999). “Twenty Years of Attacks on the RSA Cryptosystem”. *Notices of the AMS* **46**: 203-213

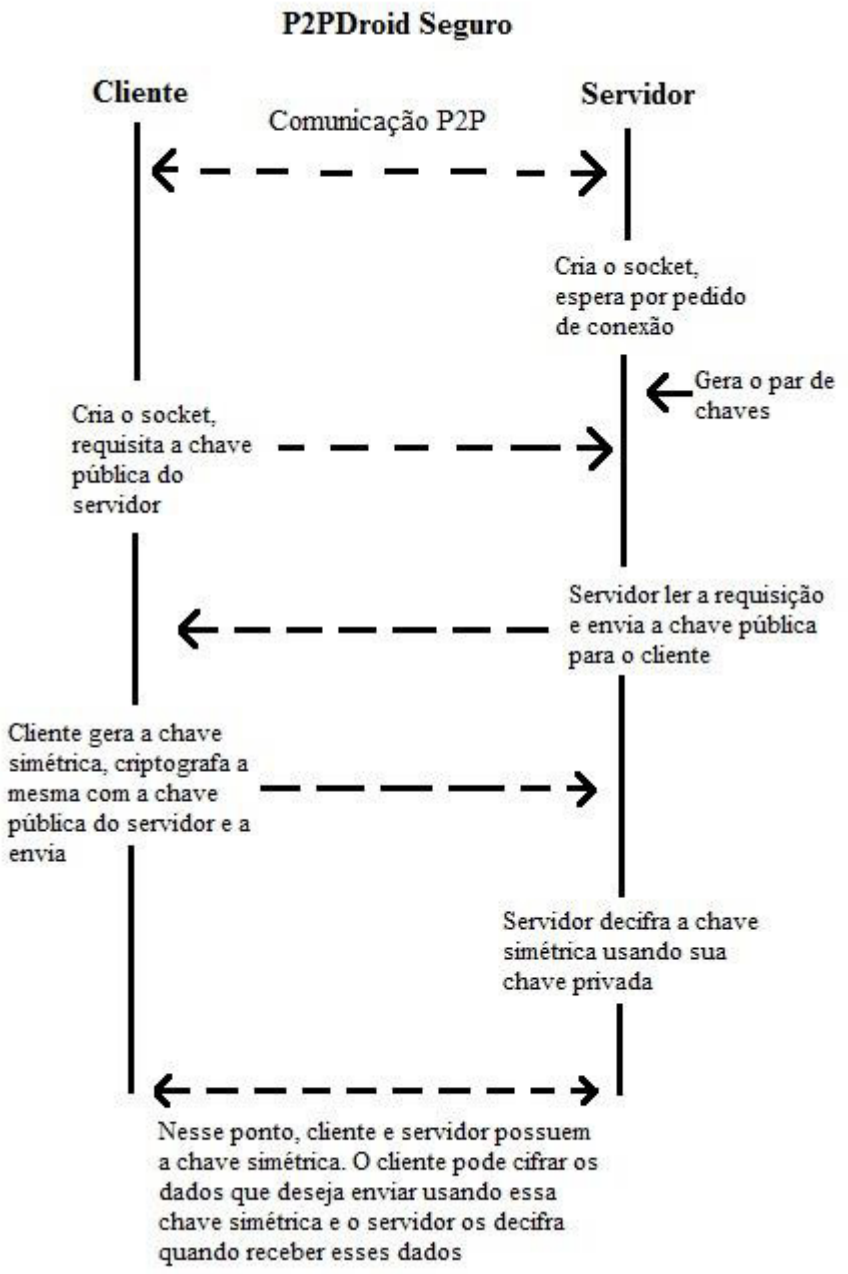
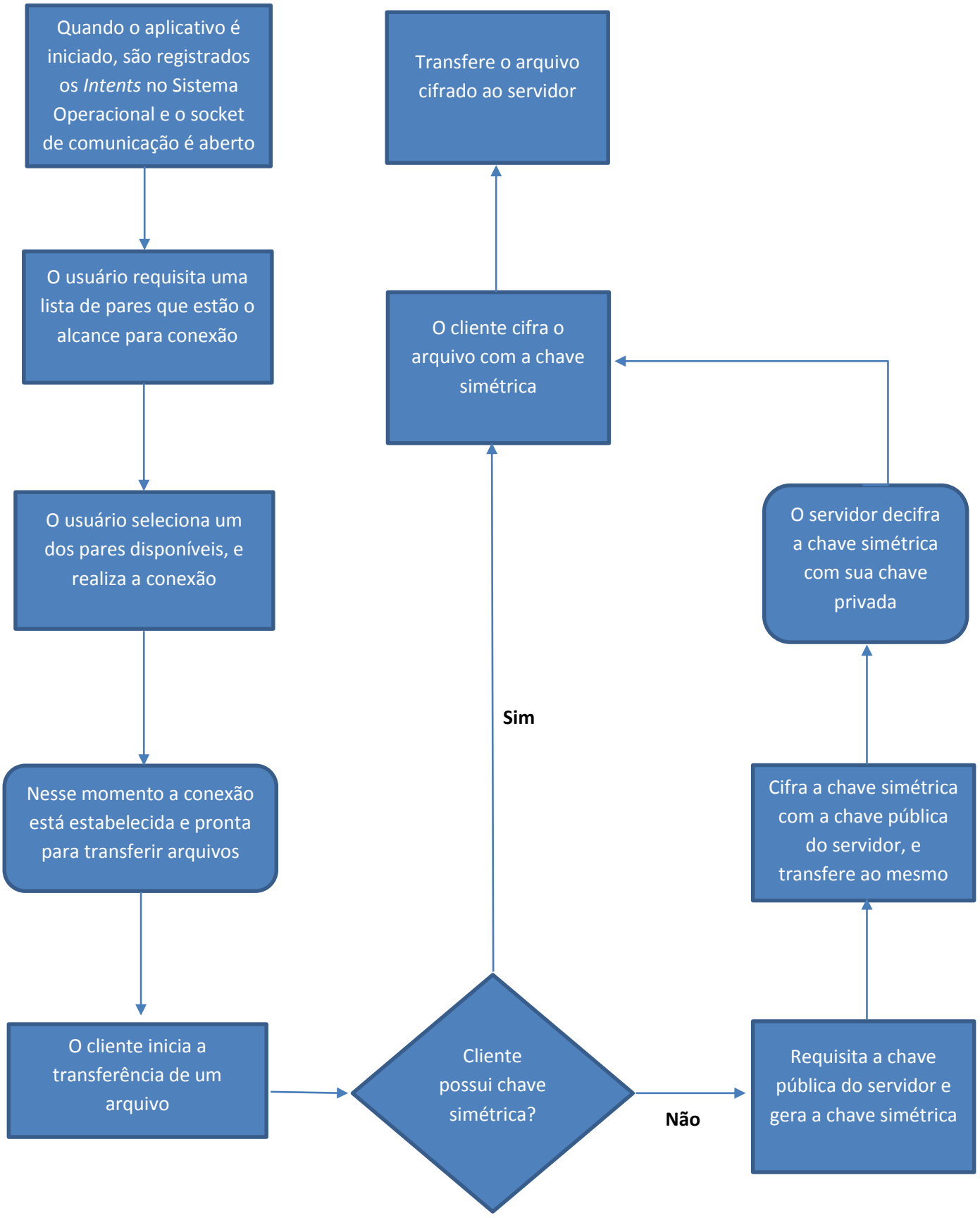


Figura 5.11 - Esquema do estabelecimento de comunicação segura do P2PDroid.

O Fluxograma 1 demonstra o funcionamento da aplicação com a camada de segurança.

Fluxograma 1 – Funcionamento do aplicativo P2PDroid



5.4.2 Resultados Obtidos

Primeiramente foi criado um arquivo de texto chamado “teste.txt” no dispositivo cliente (Figura 5.20), para que possa ser enviado ao outro dispositivo.

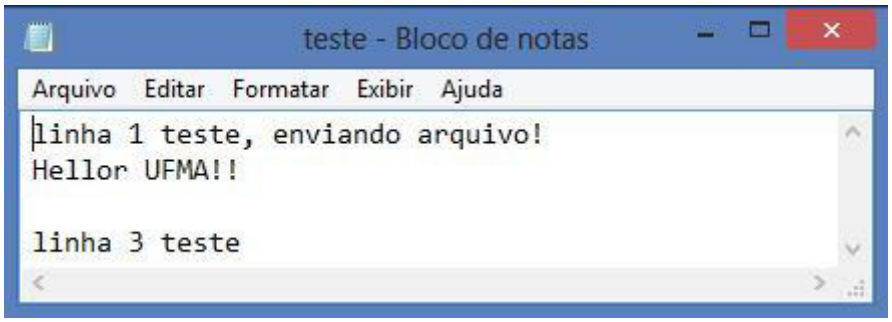


Figura 5.12 - Arquivo de teste para envio.

O primeiro teste foi realizado sem a camada de segurança, ou seja, o arquivo foi enviado pela rede sem nenhuma criptografia, podemos observar que o pacote capturado do arquivo nos mostra que podemos ler todas as informações contidas no arquivo, conforme a Figura 5.21.

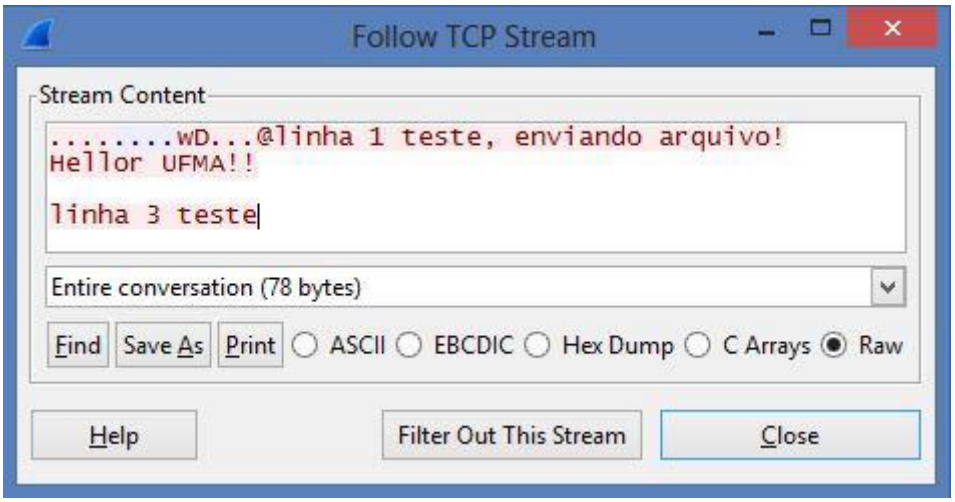


Figura 5.13 - Captura do pacote sem a camada de segurança.

No segundo teste aplicamos a camada de segurança e transferimos novamente o arquivo “teste.txt”. Na Figura 5.22 observa-se o pacote capturado durante a comunicação.

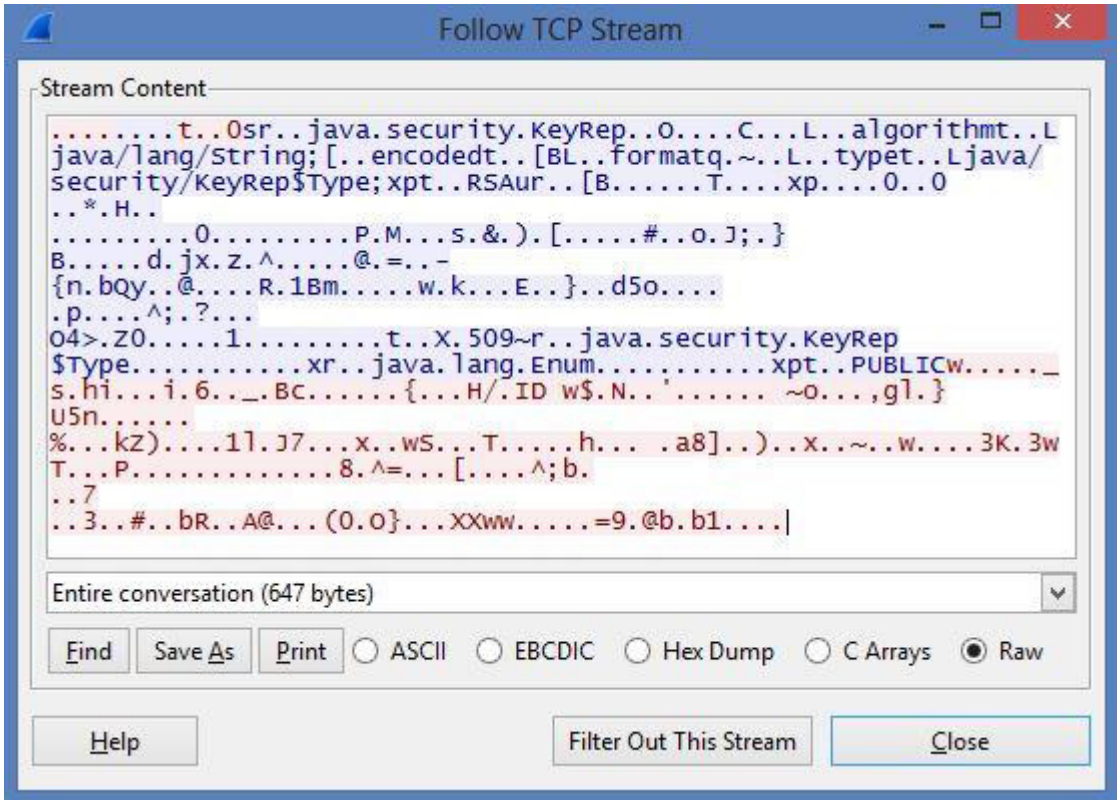


Figura 5.14 - Captura do pacote com a camada de segurança.

5.5 Considerações Finais

Na Figura 5.21, observa-se que transmitindo um arquivo de texto sem a camada de segurança desenvolvida, o tamanho da mensagem capturada foi 78 bytes, já a Figura 5.22 demonstra que usando a camada de segurança, o tamanho da mensagem salta para 647 bytes, cerca de oito vezes maior. Essa observação reforça a questão de que a criptografia deixa a comunicação mais lenta. É fácil perceber que transferir 78 bytes em uma rede sem segurança é mais rápido do que transferir 647 bytes em uma rede segura, além do fato de que é preciso um custo computacional para criptografar e descriptografar dados. Porém esse perca de velocidade em sistemas criptografados é mínima se comparada aos benefícios que a segurança de dados nos fornece.

Em suma, verificando o resultado obtido durante a fase testes; verifica-se que, aplicando a camada de segurança, tanto a cifragem quanto a decifragem dos dados funcionou perfeitamente, garantindo a segurança dos dados transmitidos na rede, assim suprindo o que foi proposto nesse trabalho.

Com uma capacidade de alcance maior, velocidade de transferência de arquivos superior e mais confiáveis, o P2PDroid tem como tendência substituir a tecnologia Bluetooth. Outros tópicos podem, e devem, ser estudados para melhorar o aplicativo; como avaliação de desempenho, custo computacional e custo de bateria. Para aumentar a velocidade de transferência, por exemplo, pode-se compactar o arquivo cifrado antes de ser enviado. Também pode-se implementar um sistema de transferência múltiplas, para o caso que ocorra transferência de arquivos grandes.

6 CONCLUSÃO

A arquitetura P2PDroid oferece serviços de compartilhamento de arquivos aos seus usuários, além de ser desenvolvido para dispositivos móveis com sistema operacional Android na versão 4.0 ou posterior, utilizando a tecnologia de *sockets* como meio de comunicação entre os dispositivos. Baseado nisso, verifica-se a necessidade de prover um trafego seguro de dados, pelo fato de que clientes P2P são bastantes visados por ataques de terceiros. Pensando nisso, foi desenvolvido uma camada de segurança que garante a integridade e confidencialidade dos dados trafegados na rede.

A dificuldade encontrada na fase de implementação da camada de segurança foi na escolha de uma metodologia para a criptografia que pudesse suprir a dinamicidade encontradas em aplicações móveis, ou seja, a camada de segurança não poderia ser dependente de algum fator externo como ocorre por exemplo em conexões seguras (SSL); onde é necessário que haja a presença de certificados digitais para que ocorra a transmissão. O objetivo é que cada aplicação seja independente e dinâmica no momento da comunicação.

A camada de segurança foi implementada usando os métodos de criptografia simétrica e assimétrica, onde a criptografia assimétrica é utilizada para a transferência da chave simétrica, e a criptografia simétrica é utilizada para a transferência dos dados em si. Assim, é garantido a confiança e velocidade dos dados trafegados, pois a criptografia simétrica trona-se mais rápida que a assimétrica e mais segura em termos de integridade dos dados para arquivos grandes. Mesmo se tratando de dispositivo móveis a arquitetura P2PDroid nos prover uma comunicação via *sockets*, o que auxiliou na implementação dessa camada.

Em trabalhos futuros, pretende-se estudar mais a fundo técnicas mais avançadas de criptografia e expandir ainda mais a camada de segurança podendo assim desenvolver outros projetos como o P2PDroid com pastas seguras e até comunicação via *sockets* seguros (SSL) de forma dinâmica.

REFERÊNCIAS

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

AROUCHA, Cláudio, Implementação de mecanismos de segurança emu ma rede peer-to-peer para dispositivos móveis: um estudo de caso da rede P2PDroid. São Luís, 2012.

BASET, S.A.; Schulzrinne, H. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol, 2004

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

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

RIGHI, R.R.; Pellissari, F.R.; Westphal, C.M. Controle de acesso e combate aos usuários caronas em sistemas peer-to-peer. Workshop de Redes Peer-to-Pee, SBRC, 2005.

SINGH, Simon. O livro dos códigos. 7. ed. Rio de Janeiro: Record, 2008.

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.

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

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

THOMAS, Stephen A, SLL & TLS Essential: Securing the Web. Ed. New York: Elsevier, 2000.

Wi-Fi Alliance, Wi-Fi Protected Setup Specification v1.0h, Dec. 2006.

Wi-Fi Alliance, P2P Technical Group, Wi-Fi Peer-to-Peer (P2P) Technical Specification v1.0, December 2009.

Site: Android Developers. Acesso em 10 de Setembro de 2013, disponível em: <http://developer.android.com/index.html>

Site: Android Open Source Project. Acesso em 20 de Setembro de 2013, disponível em: <http://source.android.com/>

Site: Comitê Gestor da Internet no Brasil (CGI.br). Acesso em 10 de Setembro de 2013, disponível em: <http://www.cgi.br/>

Site: COMODOBR, Criando Confiança na Web. Acesso em 20 de Outubro de 2013, disponível em: < http://www.comodobr.com/ssl_o_que_e.php>

Site: ConectadoESPM, Acesso em 26 de Agosto de 2013, disponível em: <http://conectadoespm.blogspot.com.br/>

Site: Copyright. Registro e depósitos de direitos de autor. Acesso em 05 de Setembro de 2013, disponível em: <http://copyright.pt/Direito-Autoral-Direito-Legal.html>

Site: DALVIKVM. Acesso em 20 de Setembro de 2013, disponível em: <http://dalvikvm.com/>

Site: DURANTE, Gabriel Barros. Redes Peer-to-Peer. Acesso em 10 de Setembro de 2013, disponível em: <http://www.gta.ufrj.br/grad/04_1/p2p/>

Sute: EDUREKA, Acesso em 19 de Setembro de 2013, disponível em: <http://www.edureka.in/blog/android-tutorials-for-beginners-activity-component/>

Site: Extensible Markup Language (XML). Acesso em 21 de Setembro de 2013, disponível em: <http://www.w3.org/XML/>

Site: Freenet, THE FREE NETWORK. Acesso em 08 de Setembro de 2013, disponível em: <https://freenetproject.org/>

Site: GARTNER INC. Acesso em 03 de Outubro de 2013, disponível em: <http://www.gartner.com/technology/home.jsp>

Site: Google. Acesso em 15 de Setembro de 2013, disponível em : < http://www.google.com/about/company/>

Site: Google Play. Acesso em 06 de Setembro de 2013, disponível em: <https://play.google.com/store>

Site: Hughes Systique Corporation, HSC. (2010). Acesso em 05 de Novembro de 2013, disponível em: <http://hsc.com/Portals/0/Uploads/Articles/WFD_Technology_Whitepaper_v_1.763503531 8321315728.pdf>

Site: Java Secure Socket Extension (JSSE). Acesso em 05 de Outubro de 2013, disponível em: <http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136007.html>

Site: Open Handset Alliance (OHA). Acesso em 03 de Setembro de 2013, disponível em: <http://www.openhandsetalliance.com/>

Site: Napster Rhapsody Company. Acesso em 05 de Setemplo de 2013, disponível em: <http://br.napster.com/start >

Site: National Institute of Standards and Technology (NIST), Acesso em 17 de outubro de 2013, disponível em: <http://www.nist.gov/index.html>

Site: National Security Agency (NSA). Acesso em 18 de Outubro de 2013, disponível em: <http://www.nsa.gov/ia/_files/NCES_WSSE_Profile_20080522.pdf>

Site: PINHEIRO, Fernando Venancio; VIEIRA, Gabriel Serafim; SILVA, Leonardo Gonçalves da. ***SSL & TLS****: Redes de Computadores I.* Acesso em 05 de Novembro de 2013, disponível em: <http://www.gta.ufrj.br/grad/11_1/tls/index.html>

Site: RFC 2246, The TLS Protocol Version 1.0. Acesso em 02 de Novembro de 2013, disponível em: < http://www.rfc-editor.org/info/rfc2246>

Site: Rivest Shamir Adleman (RSA). Laboratories. Cambridge, USA:[s.n]. Acesso em 15 de Novembro de 2013, disponível em: <http://www.rsa.com/rsalabs/>

Site: Secure Hash Algorithm (SHA-1). Acesso em 15 de Novembro de 2013, disponível em: <http://www.w3.org/TR/1998/REC-DSig-label/SHA1-1_0.html>

Site: SCREMIN, Alberto Martinez; HERRERA, Barbara Sabrina; PAIXÃO, Bianca Caruso da; TAMAKI, Daniel. Sistemas de redes peer-to-peer. Acesso em 10 de Setembro de 2013, diponível em: < http://www2.ic.uff.br/~otton/graduacao/informaticaI/P2P.pdf>

Site: Skype. Acesso em 08 de Setembro de 2013, disponível em: <http://www.skype.com/pt-br/>

Site: TECHTUDO. O que é criptografia?. Acesso em 12 de Outubro de 2013, disponível em: < http://www.techtudo.com.br/artigos/noticia/2012/06/o-que-e-criptografia.html>

Site: TRINTA, *Fernando Antonio Mota*; MACÊDO, *Rodrigo Cavalcanti de. Um estudo sobre Criptografia e Assinatura digital*. Acesso em 09 de Outubro de 2013, disponível em: <http://www.di.ufpe.br/~flash/ais98/cripto/criptografia.htm>

Site: Wi-Fi Direct in Linux, Acesso em 15 de Outubro de 2013, disponível em: <http://linuxwireless.org/en/developers/p2p/>

APÊNDICE A – Código-fonte do aplicativo P2PDroid

Arquivo ***detalhes_dispositivo.xml:***

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/details_view"
    android:orientation="horizontal"
    android:visibility="gone" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal" >

            <Button
                android:id="@+id/btn_connect"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/connect_peer_button" />

            <Button
                android:id="@+id/btn_disconnect"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/disconnect_peer_button" />

            <Button
                android:id="@+id/btn_start_gallery"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/get_file_button"
                android:visibility="gone" />
        </LinearLayout>

        <TextView
            android:id="@+id/device_address"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />

        <TextView
            android:id="@+id/device_info"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />

        <TextView
            android:id="@+id/group_owner"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />

        <TextView
            android:id="@+id/group_ip"
            android:layout_width="match_parent"
```

```
            android:layout_height="wrap_content" />
    </LinearLayout>

    <LinearLayout
        android:id="@+id/status_bar"
        android:layout_width="match_parent"
        android:layout_height="37dp"
        android:layout_gravity="bottom"
        android:layout_marginBottom="3dp"
        android:background="@android:color/background_dark"
        android:orientation="vertical" >

        <TextView
            android:id="@+id/status_text"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:layout_margin="5dp"
            android:textColor="@android:color/white" >
        </TextView>
    </LinearLayout>

</FrameLayout>
```

Arquivo ***linha_dispositivo.xml:***

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="?android:attr/listPreferredItemHeight"
    android:background="?android:attr/activatedBackgroundIndicator"
    android:padding="6dip" >

    <ImageView
        android:id="@+id/icon"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_marginRight="2dip"
        android:src="@drawable/machine" />

    <LinearLayout
        android:layout_width="0dip"
        android:layout_height="fill_parent"
        android:layout_weight="1"
        android:orientation="vertical" >

        <TextView
            android:id="@+id/device_name"
            android:layout_width="fill_parent"
            android:layout_height="0dip"
            android:layout_weight="1"
            android:gravity="center_vertical" />

        <TextView
            android:id="@+id/device_details"
            android:layout_width="fill_parent"
            android:layout_height="0dip"
            android:layout_weight="1"
```

```
            android:ellipsize="marquee"
            android:singleLine="true" />
    </LinearLayout>

</LinearLayout>
```

Arquivo ***lista_dispositivos.xml***

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingTop="3dp" >

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center_vertical"
        android:text="@string/label_me" />

    <View
        android:layout_width="fill_parent"
        android:layout_height="1dp"
        android:background="@android:color/holo_blue_light"
        android:gravity="center_vertical" />

    <!-- Self information -->

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="?android:attr/listPreferredItemHeight"
        android:background="?android:attr/activatedBackgroundIndicator"
        android:padding="3dip" >

        <ImageView
            android:id="@+id/icon"
            android:layout_width="wrap_content"
            android:layout_height="fill_parent"
            android:layout_marginRight="2dp"
            android:src="@drawable/machine" />

        <LinearLayout
            android:layout_width="0dp"
            android:layout_height="fill_parent"
            android:layout_weight="1"
            android:orientation="vertical" >

            <TextView
                android:id="@+id/my_name"
                android:layout_width="fill_parent"
                android:layout_height="0dp"
                android:layout_weight="1"
                android:gravity="center_vertical" />

            <TextView
                android:id="@+id/my_status"
                android:layout_width="fill_parent"
```

```
                android:layout_height="0dp"
                android:layout_weight="1"
                android:ellipsize="marquee"
                android:singleLine="true" />
        </LinearLayout>
    </LinearLayout>

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center_vertical"
        android:text="@string/label_peers" />

    <View
        android:layout_width="fill_parent"
        android:layout_height="1dp"
        android:background="@android:color/holo_blue_light"
        android:gravity="center_vertical" />

    <!-- Available peers -->

    <ListView
        android:id="@id/android:list"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:drawSelectorOnTop="false" />

    <TextView
        android:id="@id/android:empty"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_gravity="center"
        android:text="@string/empty_message" />

</LinearLayout>
```

Arquivo ***main.xml***

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

    <LinearLayout
        android:id="@+id/linearLayout1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >

        <fragment
            android:id="@+id/frag_list"
            android:layout_width="match_parent"
            android:layout_height="@dimen/phone_list_height"
            class="br.com.p2pdroid.ListaDispositivosFragment" >

            <!-- Preview: layout=@layout/row_devices -->
```

```
        </fragment>

        <fragment
            android:id="@+id/frag_detail"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            class="br.com.p2pdroid.DetalheDispositivoFragment" >

            <!-- Preview: layout=@layout/device_detail -->
        </fragment>
    </LinearLayout>

</LinearLayout>
```

Arquivo ***DetalheDispositivoFragment.java***

```
package br.com.p2pdroid;

import android.app.Fragment;
import android.app.ProgressDialog;
import android.content.Context;
import android.content.Intent;
import android.net.Uri;
import android.net.wifi.WpsInfo;
import android.net.wifi.p2p.WifiP2pConfig;
import android.net.wifi.p2p.WifiP2pDevice;
import android.net.wifi.p2p.WifiP2pInfo;
import android.net.wifi.p2p.WifiP2pManager.ConnectionInfoListener;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import br.com.p2pdroid.ListaDispositivosFragment.DeviceActionListener;
import com.example.android.wifidirect.R;
import crypto.Cifrador;
import java.io.File;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.security.KeyPair;
import java.security.PrivateKey;
import java.security.PublicKey;
import org.apache.commons.io.FileUtils;

public class DetalheDispositivoFragment extends Fragment implements
            ConnectionInfoListener {

      protected static final int CODIGO_RESPOSTA_ESCOLHA_ARQUIVO = 20;
      private View mContentView = null;
      private WifiP2pDevice dispositivo;
      private WifiP2pInfo informacao;
      ProgressDialog progressDialog = null;
```

```
	private static final String MIME_TYPE = "text/*";
	private static final String TIPO_ARQUIVO = ".txt";

	@Override
	public void onActivityCreated(Bundle savedInstanceState) {
		super.onActivityCreated(savedInstanceState);
	}

	@Override
	public View onCreateView(LayoutInflater inflater, ViewGroup
container,
			Bundle savedInstanceState) {

		mContentView = inflater.inflate(R.layout.detalhes_dispositivo,
null);
		mContentView.findViewById(R.id.btn_connect).setOnClickListener(
				new View.OnClickListener() {

					@Override
					public void onClick(View v) {
						WifiP2pConfig config = new
WifiP2pConfig();
						config.deviceAddress =
dispositivo.deviceAddress;
						config.wps.setup = WpsInfo.PBC;
						if (progressDialog != null
								&&
progressDialog.isShowing()) {
							progressDialog.dismiss();
						}
						progressDialog =
ProgressDialog.show(getActivity(),
								"Pressione voltar para
cancelar",
								"Connecting to :" +
dispositivo.deviceAddress,
								true, true);
						((DeviceActionListener)
getActivity()).connect(config);

					}
				});


	mContentView.findViewById(R.id.btn_disconnect).setOnClickListener(
				new View.OnClickListener() {

					@Override
					public void onClick(View v) {
						((DeviceActionListener)
getActivity()).disconnect();
					}
				});


	mContentView.findViewById(R.id.btn_start_gallery).setOnClickListener(
				new View.OnClickListener() {

					@Override
					public void onClick(View v) {
```

```
                                    Intent intent = new
Intent(Intent.ACTION_GET_CONTENT);
                                    intent.setType(MIME_TYPE);
                                    startActivityForResult(intent,

      CODIGO_RESPOSTA_ESCOLHA_ARQUIVO);
                                }
                        });

            return mContentView;
      }

      @Override
      public void onActivityResult(int requestCode, int resultCode, Intent
data) {

            Uri uri = data.getData();

            TextView statusText = (TextView) mContentView
                        .findViewById(R.id.status_text);
            statusText.setText("Enviando: " + uri);
            Log.d(P2PDroidActivity.TAG, "Intent----------- " + uri);
            Intent serviceIntent = new Intent(getActivity(),
                        FileTransferService.class);
            serviceIntent.setAction(FileTransferService.ACTION_SEND_FILE);
            serviceIntent.putExtra(FileTransferService.EXTRAS_FILE_PATH,
                        uri.toString());

      serviceIntent.putExtra(FileTransferService.EXTRAS_GROUP_OWNER_ADDRESS
,
                        informacao.groupOwnerAddress.getHostAddress());

      serviceIntent.putExtra(FileTransferService.EXTRAS_GROUP_OWNER_PORT,
                        8988);
            getActivity().startService(serviceIntent);
      }

      @Override
      public void onConnectionInfoAvailable(final WifiP2pInfo info) {
            if (progressDialog != null && progressDialog.isShowing()) {
                  progressDialog.dismiss();
            }
            this.informacao = info;
            this.getView().setVisibility(View.VISIBLE);

            TextView view = (TextView)
mContentView.findViewById(R.id.group_owner);

      view.setText(getResources().getString(R.string.group_owner_text)
                        + ((info.isGroupOwner == true) ?
getResources().getString(
                                    R.string.yes) :
getResources().getString(R.string.no)));

            view = (TextView) mContentView.findViewById(R.id.device_info);
            view.setText("IP proprietário do grupo - "
                        + info.groupOwnerAddress.getHostAddress());

            if (info.groupFormed && info.isGroupOwner) {
                  new FileServerAsyncTask(getActivity(),
```

```
mContentView.findViewById(R.id.status_text)).execute();
	} else if (info.groupFormed) {


mContentView.findViewById(R.id.btn_start_gallery).setVisibility(
				View.VISIBLE);
		((TextView) mContentView.findViewById(R.id.status_text))

.setText(getResources().getString(R.string.client_text));
	}


mContentView.findViewById(R.id.btn_connect).setVisibility(View.GONE);
}

public void verDetalhes(WifiP2pDevice dispositivo) {
	this.dispositivo = dispositivo;
	this.getView().setVisibility(View.VISIBLE);
	TextView view = (TextView) mContentView
			.findViewById(R.id.device_address);
	view.setText(dispositivo.deviceAddress);
	view = (TextView) mContentView.findViewById(R.id.device_info);
	view.setText(dispositivo.toString());

}

public void resetarViews() {

mContentView.findViewById(R.id.btn_connect).setVisibility(View.VISIBL
E);
	TextView view = (TextView) mContentView
			.findViewById(R.id.device_address);
	view.setText(R.string.empty);
	view = (TextView) mContentView.findViewById(R.id.device_info);
	view.setText(R.string.empty);
	view = (TextView) mContentView.findViewById(R.id.group_owner);
	view.setText(R.string.empty);
	view = (TextView) mContentView.findViewById(R.id.status_text);
	view.setText(R.string.empty);

mContentView.findViewById(R.id.btn_start_gallery).setVisibility(
			View.GONE);
	this.getView().setVisibility(View.GONE);
}

public static class FileServerAsyncTask extends
		AsyncTask<Void, Void, String> {

	private Context context;
	private TextView statusText;

	public FileServerAsyncTask(Context context, View statusText) {
		this.context = context;
		this.statusText = (TextView) statusText;
	}

	@Override
	protected String doInBackground(Void... params) {
		try {
			ServerSocket serverSocket = new ServerSocket(8988);
```

```
                        Log.d(P2PDroidActivity.TAG, "Servidor: Socket
aberto");
                        Socket client = serverSocket.accept();
                        Log.d(P2PDroidActivity.TAG, "Server: conexão
feita");

                        ObjectInputStream doCliente = new
ObjectInputStream(
                                    client.getInputStream());
                        ObjectOutputStream paraCliente = new
ObjectOutputStream(
                                    client.getOutputStream());

                        if (Global.CHAVE_SIMETRICA == null
                                    || Global.CHAVE_SIMETRICA.length <= 0) {

                              // gero as chaves para criptografia e
descriptografia
                              KeyPair kPair = Cifrador.geraParChaves();
                              PublicKey publicKey = kPair.getPublic();
                              PrivateKey privateKey = kPair.getPrivate();

                              String publica64 =
Cifrador.savePublicKey(publicKey);

                              paraCliente.writeObject(publica64);

                              // recebo a chave simetrica(cifrada)
                              byte[] chaveCifrada = new
byte[doCliente.readInt()];
                              doCliente.readFully(chaveCifrada);

                              Global.CHAVE_SIMETRICA =
Cifrador.decifraChaveSimetrica(
                                          privateKey, chaveCifrada);
                        }

                        // recebe o arquivo cifrado
                        byte[] arquivoCifrado = new
byte[doCliente.readInt()];
                        doCliente.readFully(arquivoCifrado);

                        // decifra o arquivo
                        byte[] arquivoDecifrado = Cifrador.decifraArquivo(
                                    arquivoCifrado, Global.CHAVE_SIMETRICA);

                        final File f = new File(

      Environment.getExternalStorageDirectory() + "/"
                                          + context.getPackageName() +
"/p2pdroidshared-"
                                          + System.currentTimeMillis()
+ TIPO_ARQUIVO);

                        File dirs = new File(f.getParent());
                        if (!dirs.exists())
                              dirs.mkdirs();
                        f.createNewFile();

                        Log.d(P2PDroidActivity.TAG,
```

```
                                    "servidor: copiando arquivo " +
f.toString());

                        FileUtils.writeByteArrayToFile(f,
arquivoDecifrado);

                        return f.getAbsolutePath();

                  } catch (Exception e) {

                  }
                  return null;
            }

            @Override
            protected void onPostExecute(String result) {
                  if (result != null) {
                        statusText.setText("Arquivo copiado - " + result);
                        Intent intent = new Intent();

      intent.setAction(android.content.Intent.ACTION_VIEW);
                        intent.setDataAndType(Uri.parse("file://" +
result), MIME_TYPE);
                        context.startActivity(intent);
                  }

            }

            @Override
            protected void onPreExecute() {
                  statusText.setText("Abrindo o socket servidor");
            }

      }
}
```

Arquivo ***FileTransferService.java***

```
package br.com.p2pdroid;

import android.app.IntentService;
import android.content.Context;
import android.content.Intent;
import android.database.Cursor;
import android.net.Uri;
import android.provider.MediaStore;
import android.util.Log;
import java.io.File;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.InetSocketAddress;
import java.net.Socket;
import java.security.PublicKey;
import org.apache.commons.io.FileUtils;
import crypto.Cifrador;

public class FileTransferService extends IntentService {
```

```
	private static final int SOCKET_TIMEOUT = 5000;
	public static final String ACTION_SEND_FILE =
"com.example.android.wifidirect.SEND_FILE";
	public static final String EXTRAS_FILE_PATH = "file_url";
	public static final String EXTRAS_GROUP_OWNER_ADDRESS = "go_host";
	public static final String EXTRAS_GROUP_OWNER_PORT = "go_port";

	public FileTransferService(String name) {
		super(name);
	}

	public FileTransferService() {
		super("FileTransferService");
	}

	@Override
	protected void onHandleIntent(Intent intent) {

		Context context = getApplicationContext();
		if (intent.getAction().equals(ACTION_SEND_FILE)) {
			String fileUri =
intent.getExtras().getString(EXTRAS_FILE_PATH);
			String host = intent.getExtras().getString(
					EXTRAS_GROUP_OWNER_ADDRESS);
			Socket socket = new Socket();
			int port =
intent.getExtras().getInt(EXTRAS_GROUP_OWNER_PORT);

			ObjectOutputStream paraServidor = null;
			ObjectInputStream doServidor = null;

			try {
				Log.d(P2PDroidActivity.TAG, "Abrindo o socket
cliente - ");
				socket.bind(null);
				socket.connect((new InetSocketAddress(host, port)),
						SOCKET_TIMEOUT);

				paraServidor = new
ObjectOutputStream(socket.getOutputStream());
				doServidor = new
ObjectInputStream(socket.getInputStream());

				// se ainda nao tiver chave simetrica, faz a troca
de chaves
				// acontece na primeira transferencia
				if (Global.CHAVE_SIMETRICA == null
						|| Global.CHAVE_SIMETRICA.length <= 0) {

					String publica = (String)
doServidor.readObject();

					PublicKey publicKey =
Cifrador.loadPublicKey(publica);

					// crio a chave simétrica(cifrada)
					byte[][] chaves =
Cifrador.gerarChaveSimetrica(publicKey);
					byte[] chaveCifrada = chaves[0];
```

```
                              // salvo a chaveDecifrada
                              Global.CHAVE_SIMETRICA = chaves[1];

                              // passo a chave simetrica cifrada para o
destinatario
                              paraServidor.writeInt(chaveCifrada.length);
                              paraServidor.write(chaveCifrada, 0,
chaveCifrada.length);
                              paraServidor.flush();
                        }

                        Uri uri = Uri.parse(fileUri);

                        File file = null;

                        if (uri.getPath().endsWith(".txt")) {
                              file = new File(fileUri.substring(7));
                        } else {
                              file = new File(getMediaPath(context, uri));
                        }

                        byte[] arquivoClaro =
FileUtils.readFileToByteArray(file);
                        byte[] arquivoCifrado =
Cifrador.cifra(arquivoClaro,
                                    Global.CHAVE_SIMETRICA);

                        paraServidor.writeInt(arquivoCifrado.length);
                        paraServidor.write(arquivoCifrado, 0,
arquivoCifrado.length);
                        paraServidor.flush();
                  } catch (Exception e) {
                        Log.e(P2PDroidActivity.TAG, e.getMessage());
                  } finally {

                        if (socket != null) {
                              if (socket.isConnected()) {
                                    try {
                                          paraServidor.close();
                                          doServidor.close();
                                          socket.close();
                                    } catch (IOException e) {
                                          e.printStackTrace();
                                    }
                              }
                        }
                  }
            }
      }

      public String getMediaPath(Context context, Uri uri) {
            String[] projection = { MediaStore.Files.FileColumns.DATA };
            Cursor cursor = context.getContentResolver().query(uri,
projection,
                        null, null, null);
            if (cursor != null) {
                  int column_index = cursor

      .getColumnIndexOrThrow(MediaStore.Files.FileColumns.DATA);
                  cursor.moveToFirst();
                  return cursor.getString(column_index);
```

```
            } else
                  return null;
      }
}
```

Arquivo ***Global.java***

```
package br.com.p2pdroid;

public class Global {
      public static byte[] CHAVE_SIMETRICA;
}
```

Arquivo ***ListaDispositivosFragment.java***

```
package br.com.p2pdroid;

import android.app.ListFragment;
import android.app.ProgressDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.net.wifi.p2p.WifiP2pConfig;
import android.net.wifi.p2p.WifiP2pDevice;
import android.net.wifi.p2p.WifiP2pDeviceList;
import android.net.wifi.p2p.WifiP2pManager.PeerListListener;
import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
import java.util.ArrayList;
import java.util.List;
import com.example.android.wifidirect.R;

public class ListaDispositivosFragment extends ListFragment implements
            PeerListListener {

      private List<WifiP2pDevice> peers = new ArrayList<WifiP2pDevice>();
      ProgressDialog progressDialog = null;
      View mContentView = null;
      private WifiP2pDevice device;

      @Override
      public void onActivityCreated(Bundle savedInstanceState) {
            super.onActivityCreated(savedInstanceState);
            this.setListAdapter(new WiFiPeerListAdapter(getActivity(),
                        R.layout.linha_dispositivo, peers));

      }

      @Override
      public View onCreateView(LayoutInflater inflater, ViewGroup
container,
                  Bundle savedInstanceState) {
```

```
            mContentView = inflater.inflate(R.layout.lista_dispositivos,
null);
            return mContentView;
      }

      public WifiP2pDevice getDevice() {
            return device;
      }

      private static String getDeviceStatus(int deviceStatus) {
            Log.d(P2PDroidActivity.TAG, "Status do par :" + deviceStatus);
            switch (deviceStatus) {
            case WifiP2pDevice.AVAILABLE:
                  return "Disponível";
            case WifiP2pDevice.INVITED:
                  return "Convidado";
            case WifiP2pDevice.CONNECTED:
                  return "Conectado";
            case WifiP2pDevice.FAILED:
                  return "Falhou";
            case WifiP2pDevice.UNAVAILABLE:
                  return "Indisponível";
            default:
                  return "Desconhecido";

            }
      }

      @Override
      public void onListItemClick(ListView l, View v, int position, long
id) {
            WifiP2pDevice device = (WifiP2pDevice)
getListAdapter().getItem(
                        position);
            ((DeviceActionListener) getActivity()).showDetails(device);
      }

      private class WiFiPeerListAdapter extends ArrayAdapter<WifiP2pDevice>
{

            private List<WifiP2pDevice> items;

            public WiFiPeerListAdapter(Context context, int
textViewResourceId,
                        List<WifiP2pDevice> objects) {
                  super(context, textViewResourceId, objects);
                  items = objects;

            }

            @Override
            public View getView(int position, View convertView, ViewGroup
parent) {
                  View v = convertView;
                  if (v == null) {
                        LayoutInflater vi = (LayoutInflater) getActivity()

      .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
                        v = vi.inflate(R.layout.linha_dispositivo, null);
                  }
                  WifiP2pDevice device = items.get(position);
```

```
                  if (device != null) {
                        TextView top = (TextView)
v.findViewById(R.id.device_name);
                        TextView bottom = (TextView) v
                                    .findViewById(R.id.device_details);
                        if (top != null) {
                              top.setText(device.deviceName);
                        }
                        if (bottom != null) {

      bottom.setText(getDeviceStatus(device.status));
                        }
                  }

                  return v;

            }
      }

      public void AtualizarEsteDispositivo(WifiP2pDevice device) {
            this.device = device;
            TextView view = (TextView)
mContentView.findViewById(R.id.my_name);
            view.setText(device.deviceName);
            view = (TextView) mContentView.findViewById(R.id.my_status);
            view.setText(getDeviceStatus(device.status));
      }

      @Override
      public void onPeersAvailable(WifiP2pDeviceList peerList) {
            if (progressDialog != null && progressDialog.isShowing()) {
                  progressDialog.dismiss();
            }
            peers.clear();
            peers.addAll(peerList.getDeviceList());
            ((WiFiPeerListAdapter)
getListAdapter()).notifyDataSetChanged();
            if (peers.size() == 0) {
                  Log.d(P2PDroidActivity.TAG, "Dispositivos não
encontrados");
                  return;
            }

      }

      public void limparPares() {
            peers.clear();
            ((WiFiPeerListAdapter)
getListAdapter()).notifyDataSetChanged();
      }

      public void onInitiateDiscovery() {
            if (progressDialog != null && progressDialog.isShowing()) {
                  progressDialog.dismiss();
            }
            progressDialog = ProgressDialog.show(getActivity(),
                        "Pressione voltar para cancelar", "procurando
pares", true,
                        true, new DialogInterface.OnCancelListener() {

                              @Override
```

```
                                public void onCancel(DialogInterface dialog) {

                                }
                        });
        }

        public interface DeviceActionListener {

                void showDetails(WifiP2pDevice device);

                void cancelDisconnect();

                void connect(WifiP2pConfig config);

                void disconnect();
        }

}
```

Arquivo ***P2PDroidActivity.java***

```
package br.com.p2pdroid;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.net.wifi.p2p.WifiP2pConfig;
import android.net.wifi.p2p.WifiP2pDevice;
import android.net.wifi.p2p.WifiP2pManager;
import android.net.wifi.p2p.WifiP2pManager.ActionListener;
import android.net.wifi.p2p.WifiP2pManager.Channel;
import android.net.wifi.p2p.WifiP2pManager.ChannelListener;
import android.os.Bundle;
import android.provider.Settings;
import android.util.Log;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.Toast;
import br.com.p2pdroid.ListaDispositivosFragment.DeviceActionListener;

import com.example.android.wifidirect.R;

public class P2PDroidActivity extends Activity implements ChannelListener,
        DeviceActionListener {

    public static final String TAG = "p2pdroid";
    private WifiP2pManager manager;
    private boolean isWifiP2pAtivo = false;
    private boolean retryChannel = false;

    private final IntentFilter intentFilter = new IntentFilter();
    private Channel channel;
    private BroadcastReceiver receiver = null;
```

```
	public void setIsWifiP2pEnabled(boolean isWifiP2pEnabled) {
		this.isWifiP2pAtivo = isWifiP2pEnabled;
	}

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.main);


	intentFilter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION);

	intentFilter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION);
		intentFilter

	.addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION);
		intentFilter

	.addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION);

		manager = (WifiP2pManager)
getSystemService(Context.WIFI_P2P_SERVICE);
		channel = manager.initialize(this, getMainLooper(), null);
	}

	@Override
	public void onResume() {
		super.onResume();
		receiver = new WiFiDirectBroadcastReceiver(manager, channel,
this);
		registerReceiver(receiver, intentFilter);
	}

	@Override
	public void onPause() {
		super.onPause();
		unregisterReceiver(receiver);
	}

	public void resetarDados() {
		ListaDispositivosFragment fragmentList =
(ListaDispositivosFragment) getFragmentManager()
				.findFragmentById(R.id.frag_list);
		DetalheDispositivoFragment fragmentDetails =
(DetalheDispositivoFragment) getFragmentManager()
				.findFragmentById(R.id.frag_detail);
		if (fragmentList != null) {
			fragmentList.limparPares();
		}
		if (fragmentDetails != null) {
			fragmentDetails.resetarViews();
		}
	}

	@Override
	public boolean onCreateOptionsMenu(Menu menu) {
		MenuInflater inflater = getMenuInflater();
		inflater.inflate(R.menu.action_items, menu);
		return true;
	}
```

```
	@Override
	public boolean onOptionsItemSelected(MenuItem item) {
		switch (item.getItemId()) {
		case R.id.atn_direct_enable:
			if (manager != null && channel != null) {
				startActivity(new
Intent(Settings.ACTION_WIRELESS_SETTINGS));
			} else {
				Log.e(TAG, "canal ou gerenciador é nulo");
			}
			return true;

		case R.id.atn_direct_discover:
			if (!isWifiP2pAtivo) {
				Toast.makeText(P2PDroidActivity.this,
R.string.p2p_off_warning,
						Toast.LENGTH_SHORT).show();
				return true;
			}
			final ListaDispositivosFragment fragment =
(ListaDispositivosFragment) getFragmentManager()
					.findFragmentById(R.id.frag_list);
			fragment.onInitiateDiscovery();
			manager.discoverPeers(channel, new
WifiP2pManager.ActionListener() {

				@Override
				public void onSuccess() {
					Toast.makeText(P2PDroidActivity.this,
							"Descoberta iniciada",
Toast.LENGTH_SHORT).show();
				}

				@Override
				public void onFailure(int reasonCode) {
					Toast.makeText(P2PDroidActivity.this,
							"Descoberta falhou : " +
reasonCode,
							Toast.LENGTH_SHORT).show();
				}
			});
			return true;
		default:
			return super.onOptionsItemSelected(item);
		}
	}

	@Override
	public void showDetails(WifiP2pDevice device) {
		DetalheDispositivoFragment fragment =
(DetalheDispositivoFragment) getFragmentManager()
				.findFragmentById(R.id.frag_detail);
		fragment.verDetalhes(device);

	}

	@Override
	public void connect(WifiP2pConfig config) {
		manager.connect(channel, config, new ActionListener() {

			@Override
```

```
                  public void onSuccess() {

                  }

                  @Override
                  public void onFailure(int reason) {
                        Toast.makeText(P2PDroidActivity.this,
                                    "Conexão falhou. Tente novamente.",
Toast.LENGTH_SHORT)
                                    .show();
                  }
            });
      }

      @Override
      public void disconnect() {
            final DetalheDispositivoFragment fragment =
(DetalheDispositivoFragment) getFragmentManager()
                        .findFragmentById(R.id.frag_detail);
            fragment.resetarViews();
            manager.removeGroup(channel, new ActionListener() {

                  @Override
                  public void onFailure(int reasonCode) {
                        Log.d(TAG, "Falha ao desconectar. Motivo :" +
reasonCode);

                  }

                  @Override
                  public void onSuccess() {
                        fragment.getView().setVisibility(View.GONE);
                  }

            });
      }

      @Override
      public void onChannelDisconnected() {
            if (manager != null && !retryChannel) {
                  Toast.makeText(this, "Canal perdido. Tentando novamente",
                              Toast.LENGTH_LONG).show();
                  resetarDados();
                  retryChannel = true;
                  manager.initialize(this, getMainLooper(), this);
            } else {
                  Toast.makeText(
                              this,
                              "Grave! Provavelmente perdeu-se o canal
pemanentemente. Tente Desabilitar/Reabilitar P2P.",
                              Toast.LENGTH_LONG).show();
            }
      }

      @Override
      public void cancelDisconnect() {
            if (manager != null) {
                  final ListaDispositivosFragment fragment =
(ListaDispositivosFragment) getFragmentManager()
                              .findFragmentById(R.id.frag_list);
                  if (fragment.getDevice() == null
```

```
                                || fragment.getDevice().status ==
WifiP2pDevice.CONNECTED) {
                        disconnect();
                    } else if (fragment.getDevice().status ==
WifiP2pDevice.AVAILABLE
                                || fragment.getDevice().status ==
WifiP2pDevice.INVITED) {

                        manager.cancelConnect(channel, new ActionListener()
{

                                @Override
                                public void onSuccess() {
                                      Toast.makeText(P2PDroidActivity.this,
                                                  "Abortando conexão",
Toast.LENGTH_SHORT).show();
                                }

                                @Override
                                public void onFailure(int reasonCode) {
                                      Toast.makeText(
                                                  P2PDroidActivity.this,
                                                  "Conexão abortada,
requisição falhou. Codgio da reação: "
                                                              + reasonCode,
Toast.LENGTH_SHORT)
                                                  .show();
                                }
                        });
                    }
            }

      }
}
```

Arquivo ***WiFiDirectBroadcastReceiver.java***

```
package br.com.p2pdroid;

import com.example.android.wifidirect.R;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.net.NetworkInfo;
import android.net.wifi.p2p.WifiP2pDevice;
import android.net.wifi.p2p.WifiP2pManager;
import android.net.wifi.p2p.WifiP2pManager.Channel;
import android.net.wifi.p2p.WifiP2pManager.PeerListListener;
import android.util.Log;

public class WiFiDirectBroadcastReceiver extends BroadcastReceiver {

      private WifiP2pManager manager;
      private Channel channel;
      private P2PDroidActivity activity;
```

```
	public WiFiDirectBroadcastReceiver(WifiP2pManager manager, Channel
channel,
			P2PDroidActivity activity) {
		super();
		this.manager = manager;
		this.channel = channel;
		this.activity = activity;
	}

	@Override
	public void onReceive(Context context, Intent intent) {
		String action = intent.getAction();
		if
(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {

			int state =
intent.getIntExtra(WifiP2pManager.EXTRA_WIFI_STATE, -1);
			if (state == WifiP2pManager.WIFI_P2P_STATE_ENABLED) {
				activity.setIsWifiP2pEnabled(true);
			} else {
				activity.setIsWifiP2pEnabled(false);
				activity.resetarDados();

			}
			Log.d(P2PDroidActivity.TAG, "P2P state changed - " +
state);
		} else if
(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {
			if (manager != null) {
				manager.requestPeers(channel, (PeerListListener)
activity

	.getFragmentManager().findFragmentById(R.id.frag_list));
			}
			Log.d(P2PDroidActivity.TAG, "P2P peers changed");
		} else if (WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION
				.equals(action)) {

			if (manager == null) {
				return;
			}

			NetworkInfo networkInfo = (NetworkInfo) intent

	.getParcelableExtra(WifiP2pManager.EXTRA_NETWORK_INFO);

			if (networkInfo.isConnected()) {

				DetalheDispositivoFragment fragment =
(DetalheDispositivoFragment) activity
						.getFragmentManager()
						.findFragmentById(R.id.frag_detail);
				manager.requestConnectionInfo(channel, fragment);
			} else {
				activity.resetarDados();
			}
		} else if (WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION
				.equals(action)) {
			ListaDispositivosFragment fragment =
(ListaDispositivosFragment) activity
```

```
      .getFragmentManager().findFragmentById(R.id.frag_list);
                  fragment.AtualizarEsteDispositivo((WifiP2pDevice) intent

      .getParcelableExtra(WifiP2pManager.EXTRA_WIFI_P2P_DEVICE));
            }
      }
}
```

Arquivo ***Cifrador.java***

```
package crypto;

import java.io.*;

import javax.crypto.*;
import javax.crypto.spec.*;
import Decoder.BASE64Decoder;
import Decoder.BASE64Encoder;
import java.security.*;
import java.security.spec.*;
import java.security.cert.*;
import java.util.Arrays;

public class Cifrador {

      private static final int RSAKEYSIZE = 1024;

      public static byte[][] gerarChaveSimetrica(PublicKey pub)
                  throws NoSuchAlgorithmException, NoSuchPaddingException,
                  InvalidKeyException, IllegalBlockSizeException,
BadPaddingException {
            // -- Gerando uma chave simétrica de 128 bits
            KeyGenerator kg = KeyGenerator.getInstance("AES");
            kg.init(128);
            SecretKey sk = kg.generateKey();
            byte[] chave_decifrada = sk.getEncoded();

            // -- Cifrando a chave com a chave pública
            Cipher rsacf = Cipher.getInstance("RSA");
            rsacf.init(Cipher.ENCRYPT_MODE, pub);
            byte[] chave_cifrada = rsacf.doFinal(chave_decifrada);

            return new byte[][] { chave_cifrada, chave_decifrada };
      }

      public static byte[] cifra(byte[] arquivoClaro, byte[]
chaveDecifrada)
                  throws NoSuchAlgorithmException, NoSuchPaddingException,
                  InvalidKeyException, IllegalBlockSizeException,
                  BadPaddingException, InvalidAlgorithmParameterException {

            byte[] textoCifrado = null;

            // -- Cifrando o texto com a chave simétrica gerada
            Cipher aescf = Cipher.getInstance("AES/CBC/PKCS5Padding");
            IvParameterSpec ivspec = new IvParameterSpec(new byte[16]);
            aescf.init(Cipher.ENCRYPT_MODE,
```

```
                        new SecretKeySpec(chaveDecifrada, "AES"), ivspec);
            textoCifrado = aescf.doFinal(arquivoClaro);

            return textoCifrado;
      }

      public static byte[] decifraChaveSimetrica(PrivateKey pvk,
                  byte[] chaveCifrada) throws NoSuchAlgorithmException,
                  NoSuchPaddingException, IllegalBlockSizeException,
                  BadPaddingException, InvalidKeyException {
            // -- Decifrando a chave simétrica com a chave privada
            Cipher rsacf = Cipher.getInstance("RSA");
            rsacf.init(Cipher.DECRYPT_MODE, pvk);
            byte[] chaveDecifrada = rsacf.doFinal(chaveCifrada);

            return chaveDecifrada;
      }

      public static byte[] decifraArquivo(byte[] arquivoCifrado,
                  byte[] chaveDecifrada) throws NoSuchAlgorithmException,
                  NoSuchPaddingException, InvalidKeyException,
                  IllegalBlockSizeException, BadPaddingException,
                  InvalidAlgorithmParameterException {

            byte[] textoDecifrado = null;
            // -- Decifrando o texto com a chave simétrica decifrada
            Cipher aescf = Cipher.getInstance("AES/CBC/PKCS5Padding");
            IvParameterSpec ivspec = new IvParameterSpec(new byte[16]);
            aescf.init(Cipher.DECRYPT_MODE,
                        new SecretKeySpec(chaveDecifrada, "AES"), ivspec);
            textoDecifrado = aescf.doFinal(arquivoCifrado);

            return textoDecifrado;
      }

      public static KeyPair geraParChaves() throws IOException,
                  NoSuchAlgorithmException,
InvalidAlgorithmParameterException,
                  CertificateException, KeyStoreException,
NoSuchProviderException {

            KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
            kpg.initialize(new RSAKeyGenParameterSpec(RSAKEYSIZE,
                        RSAKeyGenParameterSpec.F4));
            KeyPair kpr = kpg.generateKeyPair();

            return kpr;
      }

      static BASE64Decoder decoder = new BASE64Decoder();
      static BASE64Encoder encoder = new BASE64Encoder();

      public static PrivateKey loadPrivateKey(String key64) throws
IOException,
                  GeneralSecurityException {
            byte[] clear = decoder.decodeBuffer(key64);
            PKCS8EncodedKeySpec keySpec = new PKCS8EncodedKeySpec(clear);
            KeyFactory fact = KeyFactory.getInstance("RSA");
            PrivateKey priv = fact.generatePrivate(keySpec);
            Arrays.fill(clear, (byte) 0);
            return priv;
```

```
	}

	public static PublicKey loadPublicKey(String stored)
			throws GeneralSecurityException, IOException {
		byte[] data = decoder.decodeBuffer(stored);
		X509EncodedKeySpec spec = new X509EncodedKeySpec(data);
		KeyFactory fact = KeyFactory.getInstance("RSA");
		return fact.generatePublic(spec);
	}

	public static String savePrivateKey(PrivateKey priv)
			throws GeneralSecurityException {
		KeyFactory fact = KeyFactory.getInstance("RSA");
		PKCS8EncodedKeySpec spec = fact.getKeySpec(priv,
				PKCS8EncodedKeySpec.class);
		byte[] packed = spec.getEncoded();
		String key64 = encoder.encode(packed);

		Arrays.fill(packed, (byte) 0);
		return key64;
	}

	public static String savePublicKey(PublicKey publ)
			throws GeneralSecurityException {
		KeyFactory fact = KeyFactory.getInstance("RSA");
		X509EncodedKeySpec spec = fact.getKeySpec(publ,
				X509EncodedKeySpec.class);
		return encoder.encode((spec.getEncoded()));
	}
}
```