An android application which can send and receive encrypted SMS using the concepts of cryptography

Secure SMS

Jivesh Singh Gahlawat
Deepak Bhoria
Sourabh Gupta

Under the supervision of

Dr. T. Lahiri

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY – ALLAHABAD

DEOGHAT, JHALWA

ALLAHABAD- 211012, (U.P.)

INDIA

SECURE SMS

By

Jivesh Singh Gahlawat

(IIT2009008)

IIT2009008@iiita.ac.in

Sourabh Gupta

(IIT2009066)

IIT2009066@iiita.ac.in

Deepak Bhoria

(IIT2009112)

IIT2009112@iiita.ac.in

Report submitted to

Indian Institute of Information Technology, Allahabad

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, ALLAHABAD
(Deemed University)

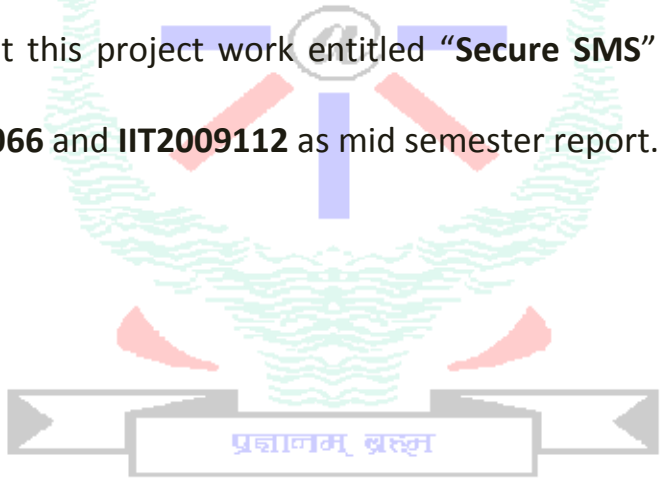
(A centre of excellence in IT, established by Ministry of HRD, Govt. of India)

Date: _____

CERTIFICATE

This is to certify that this project work entitled "**Secure SMS**" is submitted by **IIT2009008, IIT2009066** and **IIT2009112** as mid semester report.

COUNTERSIGNED



_____

Dr. T. Lahiri
(Project Supervisor)

Table of Contents

ACKNOWLEDGEMENT

As understanding of the study like this is never the outcome of the efforts of a single person, rather it bears the imprint of a number of persons who directly or indirectly helped us in completing the present study. We would be failing in our duty if we don't say a word of thanks to all those whose sincere advise make our this documentation of topic a real educative, effective and pleasurable one.

It is our privilege to study at Indian Institute of Information Technology, Allahabad where students and professors are always eager to learn new things and to make continuous improvements by providing innovative solutions. We are highly grateful to the honorable **Dr. M. D. Tiwari, Director IIIT-Allahabad,** for his ever helping attitude and encouraging us to excel in studies.

Regarding this thesis work, first and foremost, we would like to heartily thank our supervisor **Dr. T. Lahiri** for his able guidance. His fruitful suggestions, valuable comments and support were an immense help for me. In spite of his hectic schedule he took pains, with smile, in various discussions which enriched us with new enthusiasm and vigor.

ABSTRACT

In this project we are trying to build a Cryptographic Messaging application for android mobile which will send encrypted messages to the receiver using cryptographic algorithms and the receiver, who has the right key for decryption will decrypt the message.

Nowadays, due to increased technology, it is possible that a person can copy and make a duplicate SIM card from another SIM card without the awareness of the original user and then he can receive and send messages using his number and mishap can occur. Our App encrypts the message using a special key and sends the encrypted message so that even if the wrong person receives, he doesn't has the key and so would not be able to understand the message.

We have a specific type of dynamic encoding such that the key is dynamically changed so that every time it is a new key and the order of keys is also known to receiver, so even if our message reaches wrong hands and he somehow knows our key he won't be able to decode it as he doesn't knows which is the actual key we are using.

INTRODUCTION

Topic

Cryptographic Messaging is an Android Mobile Application for encrypting messages and sending them and then decrypting those messages at the receiver end.

Problem Definition

Cryptographic Messaging is an Android Mobile Application which will be using Android API's. Android is a Linux based OS provided by Google, free (for non-commercial use), that powers millions of smartphones around the world. It offers personal safety and security to smartphone users which are prone to viruses.

Even if someone uses your phone and tries to read the message, he will be asked for password to access decrypting mechanism, until then he will see encrypted message, thus our application provides both personal and public security.

Goal

Private and public security is our foremost concern. The goal of our application is to provide security by encrypting and decrypting messages and other communication text using a special encryption key or method which would be available to only the sender and receiver.

Motivation

We were motivated to do such project because when we came to know about the new misuse of increasing technology by tapping messages and misusing the "hacked" information or the leaking of personal security.

Scope

The scope of our application is huge in the present scenario as we are making this app for android and we could see that this is the era of smartphones and android being open source and the most feature full operating system for smartphones, and is also currently the largest selling smartphone os in market, makes our application of great use.

Proposed Functionalities

Below is a list of proposed functionalities in our app:

- Encrypting a message which is to be sent.
- Sending an encrypted SMS securely to any number through our application.
- On receiving the message on receiver end, it will ask the user for a password.
- If the password is authenticated, then the message will be decrypted and shown to the user for which it was intended to send.
- If a wrong password is supplied a number of times, which is customizable by user, the message will be deleted immediately.
- If the message gets deleted due to supply of incorrect password, the designated number of times, a message will be sent back to the original sender reporting the incident.
- Then depending on the sender, he may re send it or ask the receiver about the cause.

- Use of advanced features like toast notifications, autocomplete and menu scrolling in relative linear layout of our application.

Advantages over usual message sending

- The message transmission is secure and user privacy remains private.
- The message automatically gets deleted if in wrong hands.

System Requirements and Software Used

The system and software requirements for developing Android applications using the Android SDK.

Supported Operating Systems

- Windows XP (32-bit), Vista (32- or 64-bit), or Windows 7 (32- or 64-bit)
- Mac OS X 10.5.8 or later (x86 only)
- Linux (tested on Ubuntu Linux, Lucid Lynx)
    - GNU C Library (glibc) 2.7 or later is required.
    - On Ubuntu Linux, version 8.04 or later is required.
    - 64-bit distributions must be capable of running 32-bit applications.

Supported Development Environments

Eclipse IDE

- Eclipse 3.4 (Ganymede) or greater
- Eclipse JDT plugin (included in most Eclipse IDE packages)
- If you need to install or update Eclipse, you can download it from http://www.eclipse.org/downloads/.

Several types of Eclipse packages are available for each platform. For developing Android applications, we recommend that you install one of these packages:

- Eclipse IDE for Java Developers
- Eclipse Classic (versions 3.5.1 and higher)
- Eclipse IDE for Java EE Developers
- JDK 5 or JDK 6 (JRE alone is not sufficient)
- Android Development Tools plugin (recommended)

Other development environments or IDEs

- JDK 5 or JDK 6 (JRE alone is not sufficient)
- Apache Ant 1.8 or later
- **Not** compatible with Gnu Compiler for Java (gcj)

Hardware requirements

The Android SDK requires disk storage for all of the components that you choose to install. The table below provides a rough idea of the disk-space requirements to expect, based on the components that you plan to use.

Component type	Approximate size	Comments
SDK Tools	35 MB	Required.
SDK Platform-tools	6 MB	Required.
Android platform (each)	150 MB	At least one platform is required.
SDK Add-on (each)	100 MB	Optional.
USB Driver for Windows	10 MB	Optional. For Windows only.
Samples (per platform)	10M	Optional.
Offline documentation	250 MB	Optional.

Note that the disk-space requirements above are *in addition to* those of the Eclipse IDE, JDK, or other prerequisite tools that you may need to install on your development computer.

PROJECT DEVELOPMENTS

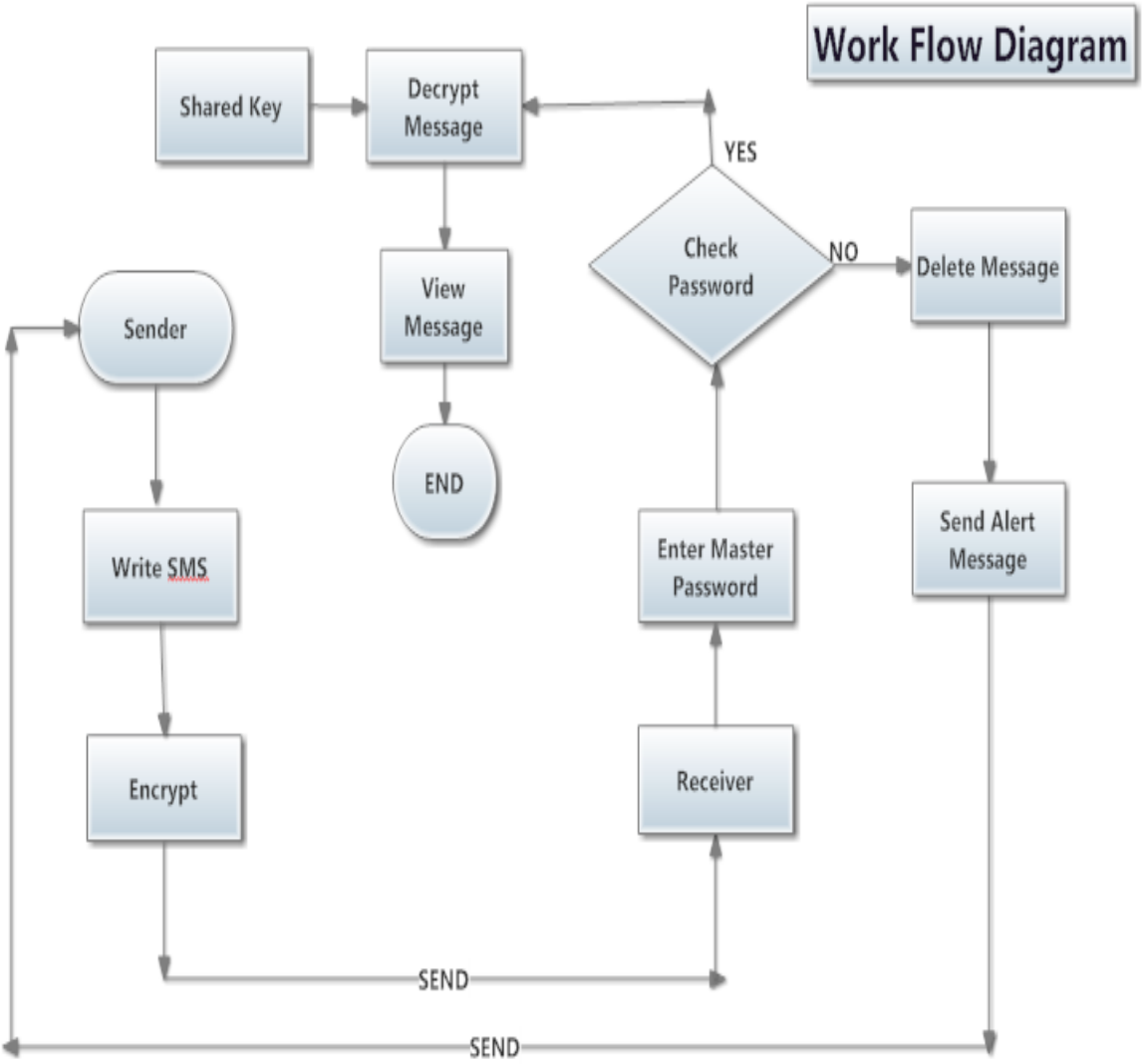
(In chronological order)

- ✓ Installation of Android SDK with needed packages, eclipse plugins, Google API and Android Virtual Device.
- ✓ Learn our way through Android development by making simple applications.
- ✓ Familiarization with Android APIs' and their usage.
- ✓ Discussed complete layout of the Secure SMS Application including work flow diagram.
- ✓ Designed the complete Graphical User Interface.
- ✓ Sent SMS message to the destination.
- ✓ Cryptography algorithms applied to make message sending secure.
- ✓ Authentication implemented to make sure no one else has access to the application.
- ✓ Key Exchange between mobiles successfully done.
- ✓ The background thread made capable of sending back confirmation from the receiver by itself.
- ✓ Toast messages installed for the user to receive the various notifications given by the app.

PROJECT TIMELINE

January	Introduction about android device and their components
February	Learning about Android API's and SDK, Java, and Eclipse IDE and implement sending and receiving SMS through our application. Learn about cryptography and how it can be applied to android app development.
March	Apply cryptography algorithms to encrypt and decrypt messages. Add authentication to view messages at receiver end.
April	Add the auto delete and back message sending capability and developing the complete 'Secure SMS' application along with GUI and testing at various stages and debugging. Final deployment of the app to android mobile.

.


Work Flow Diagram
Shared Key
Decrypt Message
YES
Check Password
NO
Delete Message
View Message
Sender
END
Write SMS
Enter Master Password
Send Alert Message
Encrypt
Receiver
SEND
SEND

LITERATURE REVIEW

Introduction to android and application development

What is Android?

Android is a software stack for mobile devices that includes an operating system, middleware and key applications. It is developed by the Open Handset Alliance led by Google. The Android SDK provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language.

Google purchased the initial developer of the software, Android Inc., in 2005. The unveiling of the Android distribution on 5 November 2007 was announced with the founding of the Open Handset Alliance, a consortium of 84 hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices. Google released most of the Android code under the Apache License, a free software license. The Android Open Source Project (AOSP) is tasked with the maintenance and further development of Android.

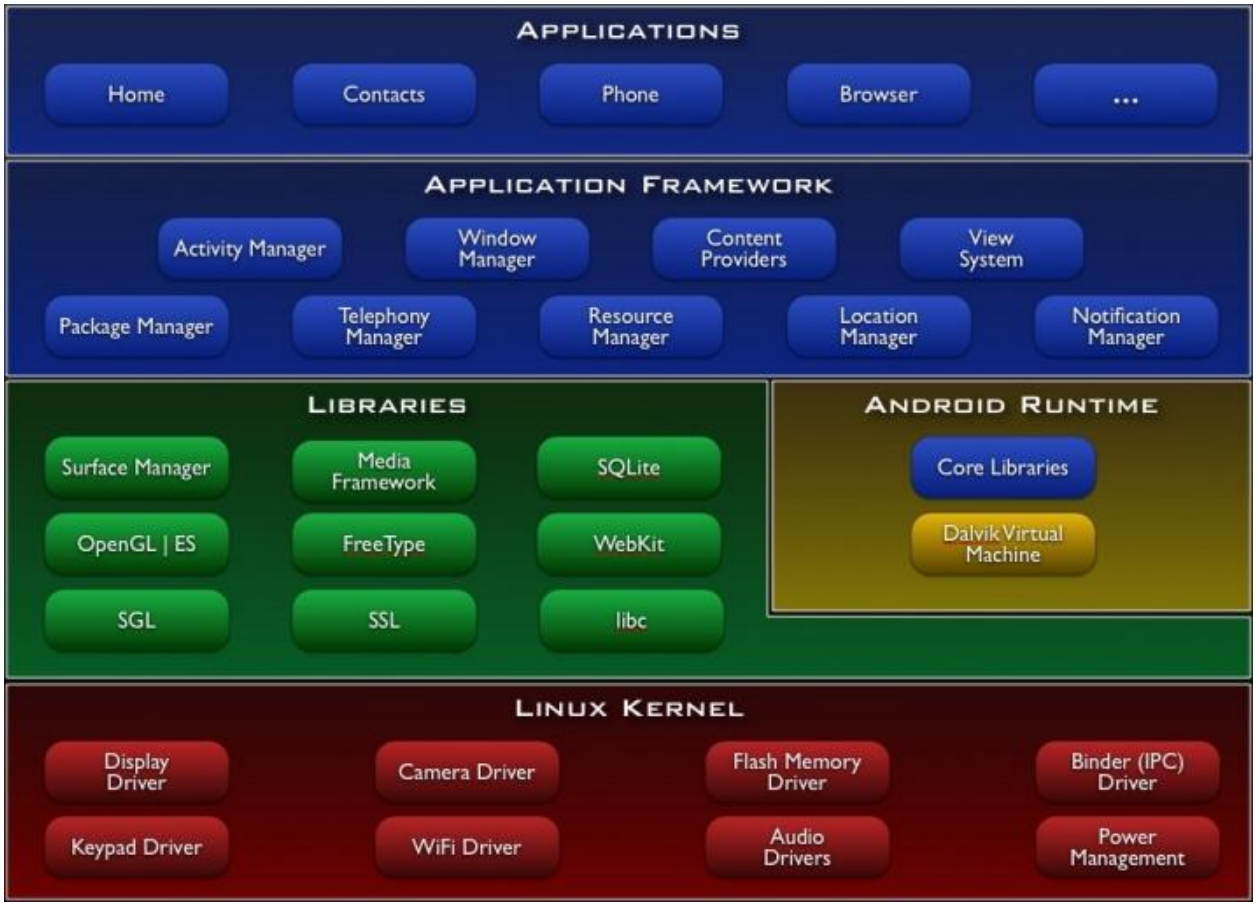
Android consists of a kernel based on the Linux kernel, with middleware, libraries and APIs written in C and application software running on an application framework which includes Java-compatible libraries based on Apache Harmony. Android uses the Dalvik virtual machine with just-in-time compilation to run compiled Java code. Android has a large community of developers writing applications ("apps") that extend the functionality of the devices. Developers write primarily in a customized version of Java. There are currently more than 250,000 apps available for Android. Apps can be downloaded from third-party sites or through online stores such as Android Play Store, the app store run by Google.

Features

Application framework: enabling reuse and replacement of components
Dalvik virtual machine: optimized for mobile devices
Integrated browser: based on the open source WebKit engine
Optimized graphics: powered by a custom 2D graphics library; 3D graphics based on the OpenGL ES 1.0 specification (hardware acceleration optional)
SQLite: for structured data storage
Media support: for common audio, video, and still image formats (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
GSM Telephony: (hardware dependent)
Bluetooth, EDGE, 3G, and WiFi: (hardware dependent)
Camera, GPS, compass, and accelerometer: (hardware dependent)
Rich development environment: including a device emulator, tools for debugging, memory and performance profiling, and a plugin for the Eclipse IDE.

Android Architecture

The following diagram shows the major components of the Android operating system. Each section is described in more detail below.



Applications

Android will ship with a set of core applications including an email client, SMS program, calendar, maps, browser, contacts, and others. All applications are written using the Java programming language.

Application Framework

By providing an open development platform, Android offers developers the ability to build extremely rich and innovative applications. Developers are free to take advantage of the device hardware, access location information, run

background services, set alarms, add notifications to the status bar, and much, much more.

Developers have full access to the same framework APIs used by the core applications. The application architecture is designed to simplify the reuse of components; any application can publish its capabilities and any other application may then make use of those capabilities (subject to security constraints enforced by the framework). This same mechanism allows components to be replaced by the user.

Underlying all applications is a set of services and systems, including:

Views: A rich and extensible set of Views that can be used to build an application, including lists, grids, text boxes, buttons, and even an embeddable web browser.

Content Providers: Content Providers that enable applications to access data from other applications (such as Contacts), or to share their own data.

Resource Manager: A Resource Manager, providing access to non-code resources such as localized strings, graphics, and layout files.

Notification Manager: A Notification Manager that enables all applications to display custom alerts in the status bar.

Activity Manager: An Activity Manager that manages the lifecycle of applications and provides a common navigation back stack.

Libraries:

Android includes a set of C/C++ libraries used by various components of the Android system. These capabilities are exposed to developers through the Android application framework. Some of the core libraries are listed below:

System C library: A BSD-derived implementation of the standard C system library (libc), tuned for embedded Linux-based devices.

Media Libraries: based on Packet Video's Open CORE; the libraries support playback and recording of many popular audio and video formats, as well as static image files, including MPEG4, H.264, MP3, AAC, AMR, JPG, and PNG.

Surface Manager: manages access to the display subsystem and seamlessly composites 2D and 3D graphic layers from multiple applications.

LibWebCore: a modern web browser engine which powers both the Android browser and an embeddable web view.
SGL: the underlying 2D graphics engine.
3D libraries: an implementation based on OpenGL ES 1.0 APIs; the libraries use either hardware 3D acceleration (where available) or the included, highly optimized 3D software rasterizer.
FreeType: bitmap and vector font rendering.
SQLite: a powerful and lightweight relational database engine available to all applications.

Android Runtime

Android includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language.

Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently. The Dalvik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint. The VM is register-based, and runs classes compiled by a Java language compiler that have been transformed into the .dex format by the included "dx" tool.

The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management.

Linux Kernel

Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.

Application Fundamentals

Android applications are written in the Java programming language. The Android SDK tools compile the code—along with any data and resource files—into an Android package, an archive file with an .apk suffix. All the code in a

single .apk file is considered to be one application and is the file that Android-powered devices use to install the application.
Once installed on a device, each Android application lives in its own security sandbox:

- The Android operating system is a multi-user Linux system in which each application is a different user.
- By default, the system assigns each application a unique Linux user ID (the ID is used only by the system and is unknown to the application). The system sets permissions for all the files in an application so that only the user ID assigned to that application can access them.
- Each process has its own virtual machine (VM), so an application's code runs in isolation from other applications.
- By default, every application runs in its own Linux process. Android starts the process when any of the application's components need to be executed, then shuts down the process when it's no longer needed or when the system must recover memory for other applications.

Application Components

Application components are the essential building blocks of an Android application. Each component is a different point through which the system can enter your application. Not all components are actual entry points for the user and some depend on each other, but each one exists as its own entity and plays a specific role—each one is a unique building block that helps define your application's overall behavior.

There are four different types of application components. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed.

On the following page, are the four types of application components are given:

Activities

An activity represents a single screen with a user interface. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Although the activities work together to form a cohesive user experience in the email application, each one is independent of the others. As such, a different application can start any one of these activities (if the email application allows it). For example, a camera application can start the activity in the email application that composes new mail, in order for the user to share a picture.

Services

A service is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it.

Content providers

A content provider manages a shared set of application data. You can store the data in the file system, an SQLite database, on the web, or any other persistent storage location your application can access. Through the content provider, other applications can query or even modify the data (if the content provider allows it). For example, the Android system provides a content provider that manages the user's contact information. As such, any application with the proper permissions can query part of the content provider (such as ContactsContract.Data) to read and write information about a particular person.

Content providers are also useful for reading and writing data that is private to your application and not shared. For example, the NotePad sample application uses a content provider to save notes.

Broadcast receivers

A broadcast receiver is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Applications can also initiate broadcasts—for example, to let other applications know that some data has been downloaded to the device and is available for them to use. Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs. More commonly, though, a broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work. For instance, it might initiate a service to perform some work based on the event.

A unique aspect of the Android system design is that any application can start another application's component. For example, if you want the user to capture a photo with the device camera, there's probably another application that does that and your application can use it, instead of developing an activity to capture a photo yourself. You don't need to incorporate or even link to the code from the camera application. Instead, you can simply start the activity in the camera application that captures a photo. When complete, the photo is even returned to your application so you can use it. To the user, it seems as if the camera is actually a part of your application.

Because the system runs each application in a separate process with file permissions that restrict access to other applications, your application cannot directly activate a component from another application. The Android system, however, can. So, to activate a component in another application, you must deliver a message to the system that specifies your intent to start a particular component. The system then activates the component for you.

The Manifest File

Before the Android system can start an application component, the system must know that the component exists by reading the application's AndroidManifest.xml file (the "manifest" file). Your application

must declare all its components in this file, which must be at the root of the application project directory.

The manifest does a number of things in addition to declaring the application's components, such as:

- Identify any user permissions the application requires, such as Internet access or read-access to the user's contacts.
- Declare the minimum API Level required by the application, based on which APIs the application uses.
- Declare hardware and software features used or required by the application, such as a camera, bluetooth services, or a multitouch screen.
- API libraries the application needs to be linked against (other than the Android framework APIs), such as the Google Maps library.

Eclipse

Eclipse is a multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system. It is written mostly in Java and can be used to develop applications in Java and, by means of various plug-ins, other programming languages including Ada, C, C++, COBOL, Perl, PHP, Python, R, Ruby (including Ruby on Rails framework) Groovy and Scheme. It can also be used to develop packages for the software Mathematica. The IDE is often called Eclipse ADT (Ada Development Toolkit) for Ada, Eclipse CDT for C/C++, Eclipse JDT for Java, and Eclipse PDT for PHP.

The initial codebase originated from VisualAge. In its default form it is meant for Java developers, consisting of the Java Development Tools (JDT). Users can extend its abilities by installing plug-ins written for the Eclipse software framework, such as development toolkits for other programming languages, and can write and contribute their own plug-in modules.

Released under the terms of the Eclipse Public License, Eclipse is free and open source software. It was one of the first IDEs to run under GNU Classpath and it runs without issues under IcedTea.

Sms Messaging Service

It sends an SMS message with user current location. We use the SmsManager class. We call the getDefault() static method to obtain an SmsManager object. The sendTextMessage() method sends the SMS message with a PendingIntent. When an SMS is sent successfully, it will display a "SMS sent" message. When it is successfully delivered, it will display a "SMS delivered" message.

GUI

GUI in android is completely XML based, so in order to be efficient in graphical layout and event handling, one must have a thorough knowledge of XML. We have made several GUI files showing various activities that are present in our application.

Android Contact List

Android Contact List is used to fetch all the phone number saved in Android Phone Directory.

For this we require these things.

1) Permissions

Add a permission to read contacts data to your application manifest.

<uses-permission android:name="android.permission.READ_CONTACTS"/>

2) Calling the Contact Picker

Within your Activity, create an Intent that asks the system to find an Activity that can perform a PICK action from the items in the Contacts URI.

Intent intent = new Intent (Intent.ACTION_PICK,
ContactsContract.Contacts.CONTENT_URI);

3) Listening for the Result

You can get the URI of the selected contact by calling getData() on the *data* Intent parameter. To get the name of the selected contact you need to use that URI to create a new query and extract the name from the returned cursor.

CRYPTOGRAPHY

Cryptography is the practice and study of techniques for secure communication in the presence of third parties (called adversaries). More generally, it is about constructing and analyzing protocols that overcome the influence of adversaries and which are related to various aspects in information security such as data confidentiality, data integrity, and authentication. Modern cryptography intersects the disciplines of mathematics, computer science, and electrical engineering. Applications of cryptography include ATM cards, computer passwords, and electronic commerce.

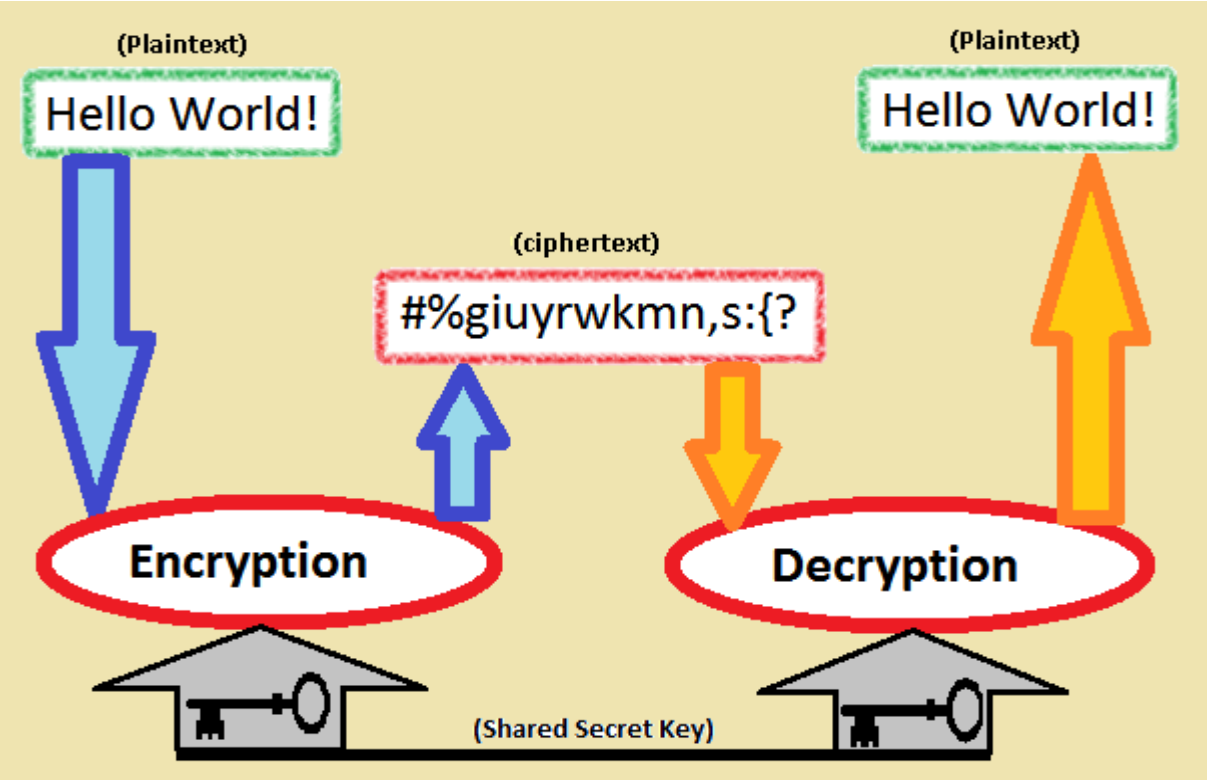
Cryptography prior to the modern age was effectively synonymous with encryption, the conversion of information from a readable state to apparent nonsense. The originator of an encrypted message shared the decoding technique needed to recover the original information only with intended recipients, thereby precluding unwanted persons to do the same. Since World War I and the advent of the computer, the methods used to carry out cryptology have become increasingly complex and its application more widespread.

Modern cryptography is heavily based on mathematical theory and computer science practice; cryptographic algorithms are designed around computational hardness assumptions, making such algorithms hard to break in practice by any adversary. It is theoretically possible to break such a system but it is infeasible to do so by any known practical means. These schemes are therefore termed computationally secure.

Until modern times cryptography referred almost exclusively to encryption, which is the process of converting ordinary information (called plaintext) into unintelligible gibberish (called ciphertext). Decryption is the reverse, in other words, moving from the unintelligible ciphertext back to plaintext. A cipher (or cypher) is a pair of algorithms that create the encryption and the reversing decryption. The detailed operation of a cipher is controlled both by the algorithm and in each instance by a "key". This is a secret parameter (ideally known only to the communicants) for a specific message exchange context. A "cryptosystem" is the ordered list of elements of finite possible plaintexts, finite possible cyphertexts, finite possible keys, and the encryption and decryption

algorithms which correspond to each key. Keys are important, as ciphers without variable keys can be trivially broken with only the knowledge of the cipher used and are therefore useless (or even counter-productive) for most purposes. Historically, ciphers were often used directly for encryption or decryption without additional procedures such as authentication or integrity checks.

In colloquial use, the term "code" is often used to mean any method of encryption or concealment of meaning. However, in cryptography, code has a more specific meaning. It means the replacement of a unit of plaintext (i.e., a meaningful word or phrase) with a code word (for example, wallaby replaces attack at dawn). Codes are no longer used in serious cryptography—except incidentally for such things as unit designations (e.g., Bronco Flight or Operation Overlord)—since properly chosen ciphers are both more practical and more secure than even the best codes and also are better adapted to computers.



HOW OUR APPLICATION IS BETTER THAN CURRENTLY EXISTING TECHNOLOGIES

In normal message sending through mobiles, due to SIM card cloning or any person using our mobile, we can't protect our personal messages to be seen by an unwanted person and thus the whole concept of privacy is abolished.

In our application, after a sender writes a message, it encrypts it using a key or an algorithm, and then sends it over the air to the receiving end. Even if someone maliciously fetches our message by using hacking or foul methods, he would not be able to decrypt it, because the key to decryption lies with the receiver. Another case would be that some person uses your device and tries to read your message, then out application will ask for access code, which after being entered a designated number of times, will delete the message and also send a report of this incident to the sender so that the sender may take appropriate action by either re sending the message or not. If it is an important information, he may better not resend it, as chances are that the cracker may be trying to crack the code of cryptography. Only our receiver side app has the right method or the right algorithm with the appropriate key to decrypt the message. Thus our technology prevents frauds and misuse of information.

CONCLUSION

Secure SMS is an android app which enables the user to send and receive SMS messages safely i.e. having controlled access over messages and who can view them.

COMPLEXITIES HANDLES BY US

REFERENCES CITED

[1] Google Android Developer [Online], http://developer.android.com/guide/index.html, January, 2012

[2] Sai Geetha, Solution Designer, Bangalore, Karnataka, India, http://saigeethamn.blogspot.com/, [Online], February, 2012

[3] Google's official blog, [Online], http://android-developers.blogspot.com/, 2012

[4] Sayed Hashimi, Satya Komatineni, Dave MacLean, Apress, "Pro Android 2", 2010

[5] Tony Hillerson, O'Reilly Media, Creativetechs, "Developing Android Applications with Java, Part 1", January, 2010

REMARKS BY BOARD MEMBERS