Mobile Application Development

Mobile application development is the process of creating software applications that run on a mobile device, and a typical mobile application utilizes a network connection to work with remote computing resources. Hence, the mobile development process involves creating installable software bundles (code, binaries, assets, etc.) , implementing backend services such as data access with an API, and testing the application on target devices.

Mobile application development is the process of creating software applications that run on a mobile device, and a typical mobile application utilizes a network connection to work with remote computing resources. Hence, the mobile development process involves creating installable software bundles (code, binaries, assets, etc.) , implementing backend services such as data access with an API, and testing the application on target devices.

Mobile application development is the process of creating software applications that run on a mobile device, and a typical mobile application utilizes a network connection to work with remote computing resources. Hence, the mobile development process involves creating installable software bundles (code, binaries, assets, etc.) , implementing backend services such as data access with an API, and testing the application on target devices.

Mobile application development is the process of creating software applications that run on a mobile device, and a typical mobile application utilizes a network connection to work with remote computing resources. Hence, the mobile development process involves creating installable software bundles (code, binaries, assets, etc.) , implementing backend services such as data access with an API, and testing the application on target devices.

Mobile Applications and Device Platforms

There are two dominant platforms in the modern smartphone market. One is the iOS platform from Apple Inc. The iOS platform is the operating system that powers Apple's popular line of iPhone smartphones. The second is Android from Google. The Android operating system is used not only by Google devices but also by many other OEMs to built their own smartphones and other smart devices.

Although there are some similarities between these two platforms when building applications, developing for iOS vs. developing for Android involves using different software development kits (SDKs) and different development toolchain. While Apple uses iOS exclusively for its own devices, Google makes Android available to other companies provided they meet specific requirements such as including certain Google applications on the devices they ship. Developers can build apps for hundreds of millions of devices by targeting both of these platforms.

Alternatives for Building Mobile Apps

There are four major development approaches when building mobile applications

- **Native Mobile Applications**
- **Cross-Platform Native Mobile Applications**
- **Hybrid Mobile Applications**
- **Progressive Web Applications**

Each of these approaches for developing mobile applications has its own set of advantages and disadvantages. When choosing the right development approach for their projects, developers consider the desired user experience, the computing resources and native features required by the app, the development budget, time targets, and resources available to maintain the app.

Native Applications

Native mobile applications are written in the programming language and frameworks provided by the platform owner and running directly on the operating system of the device such as iOS and Android.

Cross-Platform Applications

Cross-platform native mobile applications can be written in variety of different programming languages and frameworks, but they are compiled into a native application running directly on the operating system of the device.

Hybrid-Web Applications

Hybrid mobile applications are built with standard web technologies - such as JavaScript, CSS, and HTML5 - and they are bundled as app installation packages. Contrary to the native apps, hybrid apps work on a 'web container' which provides a browser runtime and a bridge for native device APIs via Apache Cordova.

Progressive Web Applications

PWAs offer an alternative approach to traditional mobile app development by skipping app store delivery and app installations. PWAs are web applications that utilize a set of browser capabilities - such as working offline, running a background process, and adding a link to the device home screen - to provide an 'app like' user experience.

Pros **Native Applications**	**Cross-Platform Applications**	**Hybrid-Web Applications**	**Progressive Web Applications**
+ Best runtime performance	+ Single code base for multiple platforms	+ Shared code base between web and mobile apps	+ Same app is available both for web and mobile
+ Direct access to device APIs	+ Easy to build and maintain your app	+ Using web development	+ No installation required, accessible through a URL

		skillset for building mobile apps	
Cons			
- Higher costs when building and maintaining your app	- Dependents on bridges and libraries for native device features	- Lower performance compared to native apps	- Limited support for native device features
-Multiple code-bases for each platform	-Performance limitations due to bridging	- Limited support for native device features	- App capabilities depend on the browser in use

Comparing Native vs. Hybrid Applications

At the highest level, there are four main ways that native apps differ from hybrid apps as illustrated in the following table.

Native	Hybrid
Platform Specific	Cross Platform
Compiled Language	Scripting / Compiled
Access to Device Hardware	Plugins / Native Modules
Platform Frameworks	Web Frameworks

Why Choose the Hybrid/Cross-platform Approach?

One problem with native mobile application development is that it requires a highly specialized skill set. Although there are large and vibrant developer communities for C and Java -- the language families that are mostly used for native development --, there are fewer developers who are knowledgeable in platform-specific versions of those languages and their respective IDEs. In fact, skilled native app developers are in such demand, that many companies are hard-pressed to hire and retain them on staff, and instead they frequently have to resort to outside 3rd party design and development houses to build their apps for them.

How Hybrid and Cross-platform Frameworks Work?

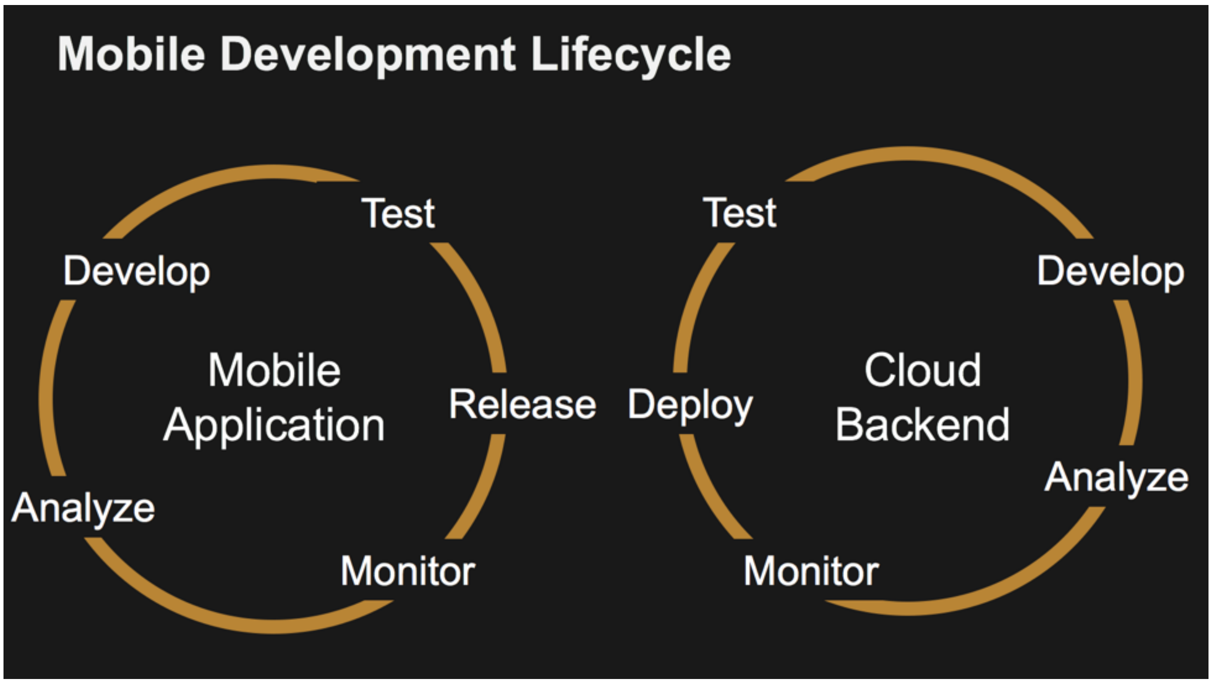
Hybrid apps allow developers to use web technologies of HTML5/CSS/JavaScript and then encapsulate those web applications in a container that allows the web application to act like a native application on the device. Since hybrid mobile apps are just web apps running on an embedded browser environment, most of the code from a web app can be used to build a mobile app. As rendering and runtime performance of mobile browsers are ever-increasing, hybrid development is a viable alternative for web developers who want to build mobile apps quickly.

Similarly, PWAs are written using traditional web application programming technologies usually including some variant of JavaScript, HTML5, and CSS, and are accessed initially through a browser on the device or computer.

Most cross-platform frameworks such as - React Native and Native Script - provides native components to work with the cross-platform code, while some others such as Flutter and Xamarin compiles cross-platform code to the native code for better performance.

The Mobile Application Development Lifecycle

There are two interlinked core components of a mobile application: 1) the mobile application “Front-End” that resides on the mobile device, and 2) the services “Back-End” that supports the mobile front-end.



Front-end vs. Back-end

In the early days of the modern smartphone applications era, mobile applications went through a similar evolution as first websites. At first, the applications and sites where wholly contained within themselves and acted as little more than static advertisements for the brand, company, product, or service.

However, as connectivity and network capabilities improved, the applications became increasingly connected to sources of data and information that lived outside of the app itself, and the apps became increasingly dynamic as they were able to update their UI and content with data received over the network from queries to data sources.

As a result, the mobile front-end applications increasingly rely on and integrated with back-end services which provide data to be consumed through the mobile front-end. Such data can include, for example, product information for e-commerce apps or flight info for travel and reservation apps. For a mobile game, the data may include new levels or challenges and scores or avatars from other players.

How Front-end 'Talks' to the Back-end?

The mobile front-end obtains the data from the back-end via a variety of service calls such as APIs. In some cases, these APIs may be owned and operated by the same entity developing the mobile application. In other cases, the API may be controlled by a third party and access is granted to the mobile application via a commercial arrangement.

For example, a developer may obtain social media or advertising content by making calls to media or advertising company services. In this case, a developer may have to sign a contract in order to obtain credentials and a key that grants access to the API and governs how that developer can use it, how much it will cost, or how frequently it may be called, or how much data can be requested over what time period.

Why Developers Use a Cloud-backend?

For most of the applications, mobile developers are responsible for creating and managing the back-end services for their application. The mobile developer may not be an expert or even particularly skilled in spinning up and running a back-end infrastructure.

In such a case, developers may prefer to take advantage of a cloud services provider -- a backend-as-a-service provider -- that handles all of the drudge work and heavy lifting of managing back-end capabilities, so the developers can focus purely on the features and functionality they are building in their app, without having to worry about scalability, security, and reliability.

The Mobile Application Front-End

The mobile front-end is the visual and interactive part of the application the user experiences. It usually resides on the device, or there is at least an icon representing the app that is visible on the home screen or is pinned in the application catalog of the device. The application can be downloaded from the platform app store, side-loaded directly onto the device, or can be reached through the device’s browser, as in the case for PWAs.

What a Front-end Development Workflow Looks Like?

When a developer says they are a mobile application programmer, they are most often referring to this front-end part of the application, and they are skilled in the languages and technologies that are used to create this front-end application.

Depending on the size of the team producing the app, there may be many different people involved in the design and development of the front-end mobile app. The team size can range from a single developer who does everything associated with building the app, to tens, hundreds, and more people with specialized skills.

For example, there may be dedicated creative/graphics designers who are responsible for creating visual elements of applications like icons, backgrounds, colors, themes and other parts of the app. The team may also have user experience and user interface designs who work on the layout of the components, how they interact with each other and the user. In the case of certain types of games, a team may include motion graphics developers and even engineers who develop engines that govern the physics of how components move in the app, like a car in a racing game.

How Mobile Aps Integrate with the Backend?

Regardless of the size of the team, a critical element of the development effort is building the app logic that is responsible for making network calls to the back-end services, retrieve data and update the data in the back-end systems with new information generated from the app.

These back-end services are typically accessed through a variety of application programming interfaces, most commonly known as APIs. There are different types of APIs, such as REST and GraphQL, and there are also a wide variety of means and styles of accessing them. While some back-end service APIs are available directly to the application through calls in the platform itself, many of the specialized services have to be integrated into the app via a software development kit, commonly known as an SDK. Once the SDK has been added to the app via the development environment, then the application can make use of the APIs defined in the SDK.

How to Interact with the Backend Data?

An example of a back-end service for a mobile front-end could be a database that contains information used in the app. To access the database directly, the mobile developer would have to know the network location of the database, the protocol for accessing the database, the credentials for authenticating and authorizing the data access, and the specific database commands needed to extract the needed data.

Alternatively, the developer can utilize a specialized API when interacting with the database; the developer may only have to know the parameters needed in a method call to get or updated the needed information. In some cases, the mobile developer may develop these APIs themselves or use the API definition provided to them by the owner/operator of the back-end resource.

Typically, a REST API is used to interact with data sources on the cloud, such as a cloud database. A GraphQL API is also another option for developers, as it makes easy to work with backend data in a mobile application. GraphQL provides querying support through a single API endpoint, and a data schema that can be used to build and easily extend data models that are used in the app.

The Mobile Application Back-End

Regardless of what front-end platform or development methodology is being used, delivering high-quality mobile applications that delight and retain users requires reliable back-end services.

Given the critical importance of back-end services for the success of the mobile application, the developers have several important architectural decisions that they must consider. These decisions include which services should they build themselves and which third party services should they leverage, and then should they run and maintain their own services or should they take advantage of 3rd party services.

The answer is increasingly clear; to improve developer productivity and efficiency, mobile app programmers should only build their own services if they are highly specific to the domain of the application and embody unique intellectual property. Also, even for the services they build themselves, they should almost always leverage cloud-based services to build and maintain their backend infrastructure.

Key Mobile Application Services

There are hundreds of cloud and 3rd party services that mobile application developers can leverage to speed up the development and delivery of their applications. However, it’s unlikely that a developer is going to be able to become an expert in each of these individual services.

Instead, the mobile developers should look for a development environment that makes it easier for them to integrate, use, and consume the most commonly required capabilities into their application quickly and easily, while still preserving the freedom to take advantage of the many individual services available.

Essential

- User Sign-up/Sign-in and Management
- Social login (Facebook sign-in, Twitter sign-in, etc.)
- Analytics and User Engagement
- Push Notifications
- Real Device Testing

Data Services

- Cloud Storage
- Real-time and Offline Data
- Application Logic/Cloud Functions

Machine Learning

- Conversational Bots
- Image and Video Recognition
- Speech Recognition

Android OS

Android OS is a Linux-based mobile operating system that primarily runs on smartphones and tablets.

The Android platform includes an operating system based upon the Linux kernel, a GUI, a web browser and end-user applications that can be downloaded. Although the initial demonstrations of Android featured a generic QWERTY smartphone and large VGA screen, the operating system was written to run on relatively inexpensive handsets with conventional numeric keypads.

Android was released under the Apache v2 open source license; this allows for many variations of the OS to be developed for other devices, such as gaming consoles and digital cameras. Android is based on open source software, but most Android devices come preinstalled with a suite of proprietary software, such as Google Maps, YouTube, Google Chrome and Gmail.

History and development

Android began its life as a Palo Alto-based startup company called Android Inc., in 2003. Originally, the company set out to develop an operating system for digital cameras, but it abandoned those efforts in lieu of reaching a broader market.

Google acquired Android Inc. and its key employees in 2005 for at least $50 million. Google marketed the early mobile platform to handset manufacturers and mobile carriers with its major benefits as flexibility and upgradability.

Google was discreetly developing Android OS when Apple released the iPhone in 2007. Previous prototypes of an Android phone closely resembled a BlackBerry, with a physical keyboard and no touchscreen. The launch of the iPhone, however, changed the mobile computing market significantly and forced Android creators to support touchscreens more heavily. Nevertheless, the HTC Dream, which was the first commercially available smartphone to run Android OS, featured a QWERTY keyboard and was met with some critical reception during its 2008 release.

In late 2007, the Open Handset Alliance (OHA) announced its formation. The OHA was a coalition of more than 30 hardware, software and telecommunications companies, including Google, Qualcomm, Broadcom, HTC, Intel, Samsung, Motorola, Sprint, Texas Instruments and Japanese wireless carriers KDDI and NTT DoCoMo. The alliance's goal was to contribute to the development of the first open source platform for mobile devices.

Google released the public beta version of Android 1.0 for developers around the same time of the alliance's announcement, in November 2007. It wasn't until Google released Android 1.5 in April 2009 that Google introduced Android's signature dessert-themed naming scheme; the name of Android 1.5 was "Cupcake." Around the time of the release of Android 4.4 KitKat, Google released an official statement to explain the naming: "Since these devices make our lives so sweet, each Android version is named after a dessert."

Android OS features

- The default UI of Android relies on direct manipulation inputs such as tapping, swiping and pinching to initiate actions. The device provides haptic feedback to the user via alerts such as vibrations to respond to actions. If a user presses a navigation button, for example, the device vibrates.
- When a user boots a device, Android OS displays the home screen, which is the primary navigation hub for Android devices and is comprised of widgets and app icons. Widgets are informational displays that automatically update content such as weather or news. The home screen display can differ based on the device manufacturer that is running the OS. Users can also choose different themes for the home screen via third-party apps on Google Play.
- A status bar at the top of the home screen displays information about the device and its connectivity, such as the Wi-Fi network that the device is connected to or signal strength. Users can pull down the status bar with a swipe of a finger to view a notification screen.

- Android OS also includes features to save battery usage. The OS suspends applications that aren't in use to conserve battery power and CPU usage. Android includes memory management features that automatically close inactive processes stored in its memory.
- Android runs on both of the most widely deployed cellular standards, GSM/HSDPA and CDMA/EV-DO.

Android also supports:

- Bluetooth
- Edge
- 3G communication protocols, like EV-DO and HSDPA
- Wi-Fi
- Autocorrect
- SMS and MMS messaging
- video/still digital cameras
- GPS
- compasses
- accelerometers
- accelerated 3D graphics
- multitasking applications

Android OS versions

Google makes incremental changes to the OS with each release. This often includes security patches and performance improvements.

- **Android 1.0.** Released Sept. 23, 2008. Included a suite of Google apps, including Gmail, Maps, Calendar and YouTube.
- **Android 1.5 (Cupcake).** Released April 27, 2009. Introduced an onscreen virtual keyboard and the framework for third-party app widgets.
- **Android 1.6 (Donut).** Released Sept. 15, 2009. Introduced the ability for the OS to run on different screen sizes and resolutions; added support for CDMA networks.
- **Android 2.0 (Eclair).** Released Oct. 26, 2009. Added turn-by-turn voice navigation, real-time traffic information, pinch-to-zoom capability.
- **Android 2.2 (Froyo).** Released May 20, 2010. Added dock at the bottom of the home screen and voice actions, which allows users to tap an icon and speak a command. Also introduced support for Flash to the web browser.
- **Android 2.3 (Gingerbread).** Released Dec. 6, 2010. Introduced black and green into the UI.

- **Android 3.0 to 3.2 (Honeycomb).** Released Feb. 22, 2011. This release was exclusive to tablets and introduced a blue, space-themed holographic design.
- **Android 4.0 (Ice Cream Sandwich).** Released Oct. 18, 2011. Introduced a unified UI to both tablets and smartphones; emphasized swiping as a navigational method.
- **Android 4.1 to 4.3 (Jelly Bean).** Released July 9, 2012, Nov. 13, 2012, and July 24, 2013, respectively. Introduced Google Now, a day planner service. Added interactive notifications and improved voice search system.
- **Android 4.4 (KitKat).** Released Oct. 31, 2013. Introduced lighter colors into the UI, along with a transparent status bar and white icons.
- **Android 5.0 (Lollipop).** Released Nov. 12, 2014. Incorporated a card-based appearance in the design with elements such as notifications and Recent Apps list. Introduced hands-free voice control with the spoken "OK, Google" command.
- **Android 6.0 (Marshmallow).** Released Oct. 5, 2015. This release marked Google's adoption of an annual release schedule. Introduced more granular app permissions and support for USB-C and fingerprint readers.
- **Android 7.0 and 7.1 (Nougat).** Released Aug. 22, 2016 and Oct. 4, 2016, respectively. Introduced a native split-screen mode and the ability to bundle notifications by app.
- **Android 8.0 and 8.1 (Oreo).** Released Aug. 21, 2017 and Dec. 5, 2017, respectively. These versions introduced a native picture-in-picture (PIP) mode and the ability to snooze notifications. Oreo was the first version to incorporate Project Treble, an effort by OEMs to provide more standardized software updates.
- **Android 9.0 (Pie).** Released Aug. 6, 2018. This version replaced Back, Home and Overview buttons for a multifunctional Home button and a smaller Back button. Introduced productivity features, including suggested replies for messages and brightness management capabilities.
- **Android 10 (Android Q).** Released Sept. 3, 2019. Abandoned the Back button in favor of a swipe-based approach to navigation. Introduced a dark theme and Focus Mode, which enables users to limit distractions from certain apps.
- **Android 11 (Red Velvet Cake).** Released Sept. 8, 2020. Added built-in screen recording. Created a single location to view and respond to conversations across multiple messaging apps. This version also updated the chat bubbles so users can pin conversations to the top of apps and screens.
- **Android 12 (Snow Cone).** Released Oct. 4, 2021. Added customization options for the user interface. The conversation widget let users store preferred contacts on their home screens. Added more privacy options, including sharing when apps access information such as camera, photos or microphone.
- **Android 12L.** Released March 7, 2022.The L stands for larger screens. This update aimed to improve the user interface and optimize for the larger screen of a tablet, foldable or Chromebook. This update added a dual-panel notification center for tablets and foldables.

- **Android 13 (Tiramisu).** Released Aug. 15, 2022. Included more customizable options including color, theme, language and music. Security updates included control over information apps can access, notification permission required for all apps and clearing of personal information on clipboard. This update enables multitasking by sharing of messages, chats, links and photos across multiple Android devices -- including phones, tablets and Chromebooks.

Hardware

Android uses ARM for its hardware platform; later versions of Android OS support x86 and x86-64 architectures. Starting in 2012, device manufacturers released Android smartphones and tablets with Intel processors.

The minimum hardware requirements of Android depend on the device's screen size and CPU type and density. Originally, Google required a 200 MHz processor, 32 MB of storage and 32 MB of RAM.

Google releases documentation with hardware requirements that original equipment manufacturers (OEMs) must meet for a device to be "Google Approved," which means that it will ship with official Google apps. The open source nature of Android, however, means that it can also run on lesser hardware, and vice versa.

Comparisons with other mobile OSes

Initially, Android's creators believed that the OS would compete with other mobile operating systems such as Symbian and Microsoft Windows Mobile.

Symbian was a closed OS with a microkernel and a UI that provided the graphical shell. Many mobile manufacturers used Symbian OS, including Nokia, Samsung and Motorola. Symbian was a popular OS worldwide, but it did not gain major popularity in North America. Symbian's design was not as simple as Android and iOS, however, and the OS was difficult to program. Symbian OS development was discontinued in 2014.

Windows Mobile originated from Windows CE, an embedded OS, and first appeared on a Pocket PC 2000. Microsoft marketed the mobile OS toward businesses. Competition from Android and iOS forced Microsoft to make changes; the company replaced Windows Mobile with Windows Phone in 2010, aimed at the consumer market. Microsoft phased out Windows Phone in favor of Windows 10 Mobile, but that OS was also discontinued; Microsoft declared its end of life for Jan. 14, 2020.

Android's main competitor is Apple iOS. Both iOS and Android OS offer comparative features. Apple iOS is a proprietary OS with a fixed interface, whereas Android is an open source OS that offers more flexibility and customization.

Android's global market share in 2022 was 71.85%, according to a Statcounter report. Apple iOS' global market share was 27.5%. In the U.S., however, Apple dominates the market share at 55.25%; Android claims 44.43%, followed by Samsung at .27% and Windows at .02%.

Criticism

The most significant user criticism of Android is that the OS is fragmented. The flexible, open source nature of Android results in many variations of hardware and software. Many devices run older versions of Android. As of July 2022, 29.63% of Android users run version 11 operating system, 21.8% run version 10, 20.86% use version 12 and 10.74% use version 9, according to Statcounter.

Device fragmentation creates challenges for developers because it's difficult to develop apps that work across all device types and versions. Fragmentation is also a problem for businesses; IT staff cannot easily secure and manage devices that run on a variety of hardware and software. Google launched Project Treble as a potential solution to this problem. The initiative separates the Android OS from OEM modifications and enables software updates to be deployed faster.

Another criticism of Android OS is that Android applications can be easily pirated. With the release of Android Jelly Bean, however, Google offered the ability for developers to encrypt paid applications.

Understanding Android System Architecture

Android is comprised of several mechanisms playing a role in security checking and enforcement. Like any modern operating system, many of these mechanisms interact with each other, exchanging information about subjects (apps/users), objects(other apps, files, devices),and operations to be performed (read, write, delete, and so on). Oftentimes, enforcement occurs without incident; but occasionally, things slip through the cracks, affording opportunity for abuse. This chapter discusses the security design and architecture of Android, setting the stage for analysing the overall attack surface of the Android platform.
The general Android architecture has, at times, been described as “Java on Linux.” However, this is a bit of a misnomer and doesn’t entirely do justice to the complexity and architecture of the platform. The overall architecture consists of components that fall into five main layers, including Android applications, the Android Framework, the Dalvik virtual machine, user-space native code, and the Linux kernel. Figure 2-1 shows how these layers comprise the Android software stack.

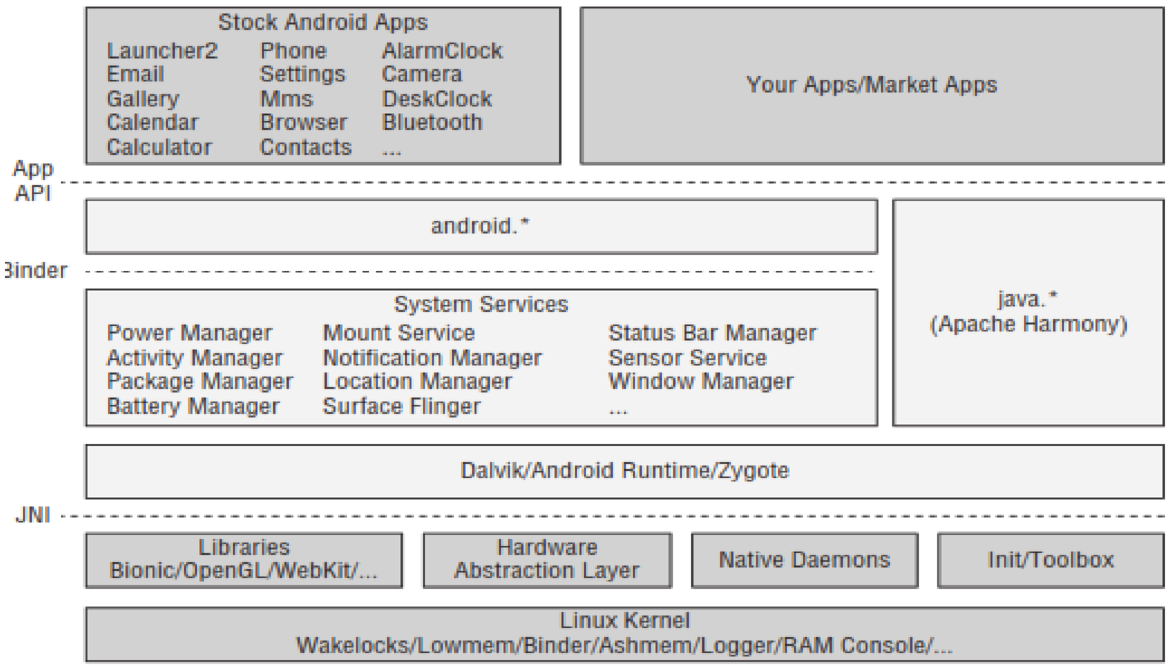
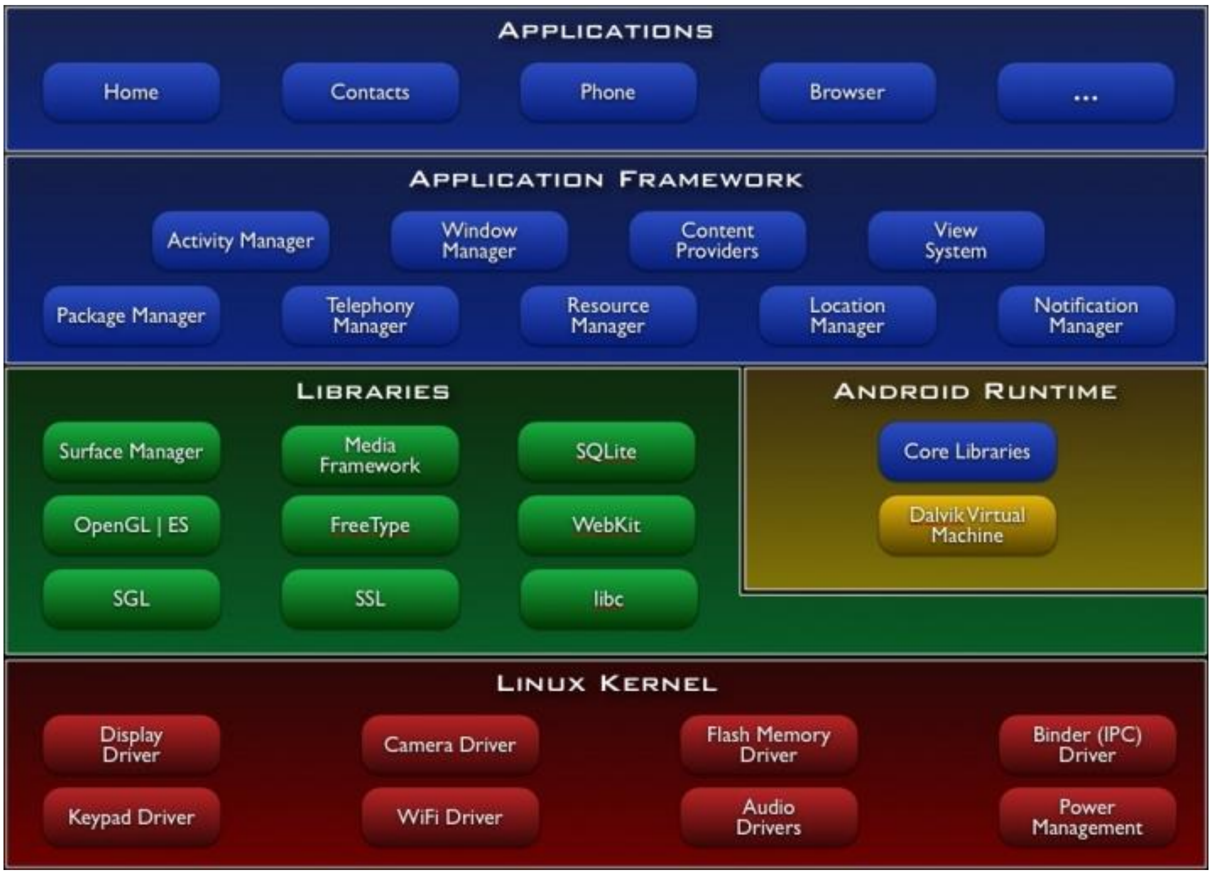


Figure 2-1: General Android system architecture



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

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:

- 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 that enable applications to access data from other applications (such as Contacts), or to share their own data
- A Resource Manager, providing access to non-code resources such as localized strings, graphics, and layout files
- A Notification Manager that enables all applications to display custom alerts in the status bar
- An Activity Manager that manages the lifecycle of applications and provides a common navigation backstack

-

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 PacketVideo's OpenCORE; 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.

iOS

IOS stands for iPhone operating system. It is a proprietary mobile operating system of Apple for its handheld. It supports Objective-C, C, C++, Swift programming language. It is based on the Macintosh OS X. After Android, it is the world's second most popular mobile operating system. Many of Apple's mobile devices, including the iPhone, iPad, and iPod, run on this operating system. To control the device, iOS employs a multi-touch interface, such as sliding your finger across the screen to advance to the next page or pinching your fingers to zoom in or out of the screen.

Features of IOS Platform

iOS has become popular because of its prominent features. The following are the popular features of iOS. Let’s get into details.

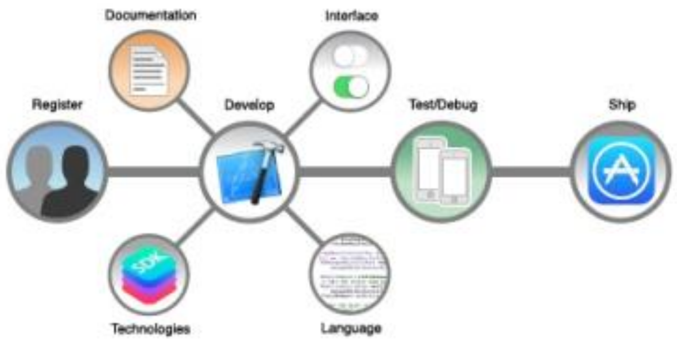
1. **Multitasking:** iPhone offers multitasking features. It started with iPhone 4, iPhone 3GS. By using the multitasking feature on an iOS device or using a multi-finger gesture on an iPad, you can swiftly go from one app to another at any moment.
2. **Social Media:** Sharing content and displaying an activity stream are just a few of the ways iOS makes it simple to integrate social network interactions into the app.
3. **iCloud:** Apple’s iCloud is a service that offers Internet-based data storage. It works on all Apple devices and has some Windows compatibility, and handles most operations in the background. It is highly encrypted. It offers the backup option to help the user not lose any of their data.
4. **In-App purchase:** In-app purchases, which are available on all Apple platforms, provide users with additional material and services, such as digital items(iOS, iPadOS, macOS, watchOS), subscriptions, and premium content, right within the app. You may even use the App Store to promote and sell in-app purchases.
5. **Game Center:** Game Center, Apple’s social gaming network, adds even more pleasure and connection to your games. Game Center provides access to features such as leaderboards, achievements, multiplayer capability, a dashboard, and more.
6. **Notification Center:** Notification Center is a feature in iOS that shows you all of your app alerts in one place. Rather than needing immediate resolution, it displays notifications until the user completes an associated action. However, we can control the notification settings.
7. **Accelerometer:** An accelerometer is a device that detects changes in velocity along a single axis. A three-axis accelerometer is built into every iOS device, providing acceleration readings in each of the three axes. LIS302DL 3-axis MEMS-based accelerometer is used in the original iPhone and first-generation iPod touch.
8. **Gyroscope:** The rate at which a gadget rotates around a spatial axis is measured using a gyroscope. A three-axis gyroscope is found in many iOS devices, and it provides rotation data in each of the three axes.
9. **GPS:** To detect your location, the iPhone uses an inbuilt Assisted GPS (AGPS) chip. You don’t even need to install this function because it’s already integrated into your iPhone. As it provides an approximation of your location based on satellite information, this system is faster than standard GPS.
10. **Accessibility:** Every Apple product and service is built with one-tap accessibility capabilities that work the way you do.
11. **Bluetooth:** Apple supplies the Core Bluetooth framework, which includes classes for connecting to Bluetooth-enabled low-energy wireless technology.

12. **Orientations:** The iOS apps can be used in both portrait and landscape modes. Apple, on the other hand, provides size classes in XCode for creating interfaces in landscape and portrait orientations.
13. **Camera integration:** In iOS, Apple provides the AVFoundation Capture Subsystem, which is a standard high-level architecture for audio, image, and video capture.
14. **Location services:** The Location Services enable applications and websites to access the user's device location with the user's permission. When location services are operational, the status bar displays a black or white arrow icon.
15. **Maps:** Apple offers an online mapping service that can be utilized as the iOS default map system. It has a variety of functions, such as a flyover mode. Apple's MapKit may be used to create applications that utilize maps.

History

The iPhone was first released in June 2007 and on September 5, 2007, Apple released the iPod Touch which had most of the non-phone abilities of the iPhone. In June 2010 Apple rebranded iPhone Os as iOS. iPad first generation iPad was released in April 2010 and the iPad Mini was released in November 2012.

To develop iOS application use the following steps:



- Register as a developer.
- Design beautiful app interfaces.
- Learn the Objective-C language.
- Develop great apps.
- Learn about the technologies available to you.
- Access the documentation.
- Debug and test your app.
- Ship your app.

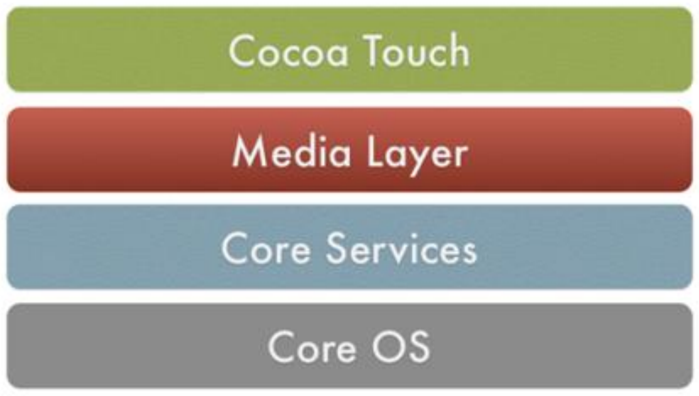
Architecture of iOS

Architecture of IOS is a layered architecture. At the uppermost level iOS works as an intermediary between the underlying hardware and the apps you make. Apps do not communicate to the underlying hardware directly.

Apps talk with the hardware through a collection of well defined system interfaces. These interfaces make it simple to write apps that work constantly on devices having various hardware abilities.

Lower layers gives the basic services which all application relies on and higher level layer gives sophisticated graphics and interface related services.

Apple provides most of its system interfaces in special packages called frameworks. A framework is a directory that holds a dynamic shared library that is .a files, related resources like as header files, images, and helper apps required to support that library. Every layer have a set of Framework which the developer use to construct the applications.



1.CoreOSLayer:
The Core OS layer holds the low-level features that most other technologies are built upon.

- Core Bluetooth Framework.
- Accelerate Framework.
- External Accessory Framework.
- Security Services framework.
- Local Authentication framework.

64-Bit support from IOS7 supports the 64 bit app development and enables the application to run faster.

2.CoreServicesLayer
Some of the Important Frameworks available in the core services layers are detailed:

- **Address book framework** – Gives programmatic access to a contacts database of user.
- **Cloud Kit framework** – Gives a medium for moving data between your app and iCloud.

- **Core data Framework** – Technology for managing the data model of a Model View Controller app.
- **Core Foundation framework –** Interfaces that gives fundamental data management and service features for iOS apps.
- **Core Location framework –** Gives location and heading information to apps.
- **Core Motion Framework –** Access all motion based data available on a device. Using this core motion framework Accelerometer based information can be accessed.
- **Foundation Framework –** Objective C covering too many of the features found in the Core Foundation framework
- **Healthkit framework –** New framework for handling health-related information of user
- **Homekit framework** – New framework for talking with and controlling connected devices in a user's home.
- **Social framework –** Simple interface for accessing the user's social media accounts.
- **StoreKit framework –** Gives support for the buying of content and services from inside your iOS apps, a feature known asIn-App Purchase.

3. **Media Layer:** Graphics, Audio and Video technology is enabled using the Media Layer. **Graphics Framework**:

- **UIKit Graphics –** It describes high level support for designing images and also used for animating the content of your views.
- **Core Graphics framework** – It is the native drawing engine for iOS apps and gives support for custom 2D vector and image based rendering.
- **Core Animation –** It is an initial technology that optimizes the animation experience of your apps.
- **Core Images –** gives advanced support for controling video and motionless images in a nondestructive way
- **OpenGl ES and GLKit** – manages advanced 2D and 3D rendering by hardware accelerated interfaces
- **Metal –** It permits very high performance for your sophisticated graphics rendering and computation works. It offers very low overhead access to the A7 GPU.

Audio Framework:

- **Media Player Framework –** It is a high level framework which gives simple use to a user's iTunes library and support for playing playlists.
- **AV Foundation –** It is an Objective C interface for handling the recording and playback of audio and video.
- **OpenAL –** is an industry standard technology for providing audio.

Video Framework

- **AV Kit –** framework gives a collection of easy to use interfaces for presenting video.
- **AV Foundation –** gives advanced video playback and recording capability.
- **Core Media –** framework describes the low level interfaces and data types for operating media.

Cocoa Touch Layer

- **EventKit framework –** gives view controllers for showing the standard system interfaces for seeing and altering calendar related events
- **GameKit Framework –** implements support for Game Center which allows users share their game related information online
- **iAd Framework –** allows you deliver banner-based advertisements from your app.
- **MapKit Framework –** gives a scrollable map that you can include into your user interface of app.
- **PushKitFramework –** provides registration support for VoIP apps.
- **Twitter Framework –** supports a UI for generating tweets and support for creating URLs to access the Twitter service.
- **UIKit Framework –** gives vital infrastructure for applying graphical, event-driven apps in iOS. Some of the Important functions of UI Kit framework:

-Multitasking support.
– Basic app management and infrastructure.
– User interface management
– Support for Touch and Motion event.
– Cut, copy and paste support and many more.

iOS System Security

System security is central to security in iOS. It makes the hardware and software securely integrated with each other such that every component in iOS is secure and trusted. Fig.1 is a high-level overview of iOS security architecture, the details are explained in the following sections.

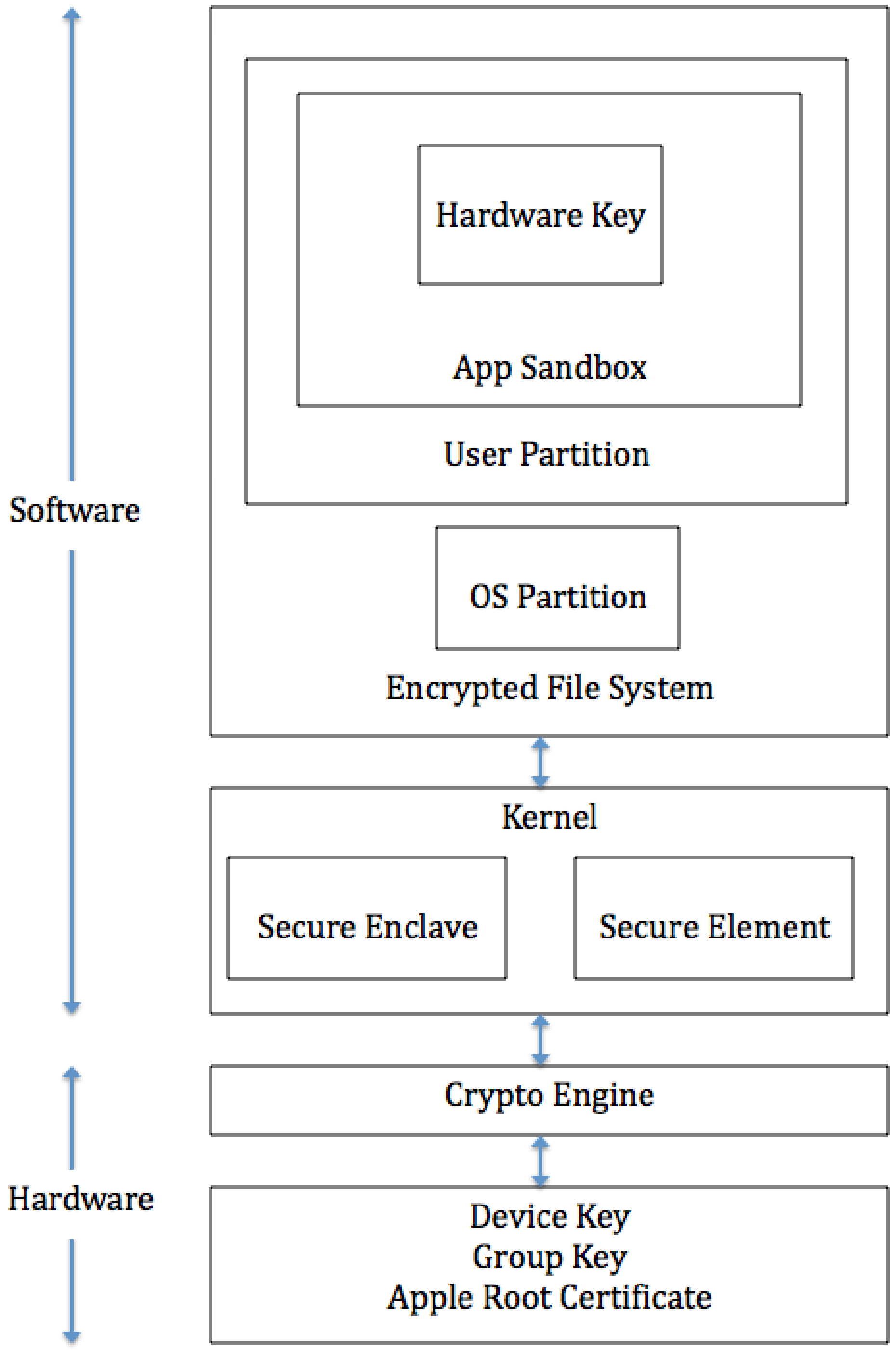


Fig.1 Security architecture diagram of iOS

2.1 Secure booting process

During the booting process, iOS uses a mechanism called "secure boot chain" to ensure that the low-level software is not compromised and iOS is running on a validated iOS device. Each step in secure boot chain verifies if the next step of chain is valid and signed by Apple. The booting process will only proceed to the next step of chain if the verification succeeds.

When you turn on an iOS device, the processor first executes the code from Boot ROM (read-only-memory). The code in Boot ROM is created during chip fabrication, hence it is trusted and immutable. The code in Boot ROM also contains the Apple Root CA public key, which will be used to verify if Low-Level Bootloader (LLB) is signed by Apple. If LLB is valid, the processor will run the next-stage bootloader, iBoot, which will in turn verify and run the iOS kernel.

2.2 Secure Enclave

The Secure Enclave is a coprocessor for Apple’s A-series processor. It has its own secure boot separated from the application processor, communication between it and the application processor is highly encapsulated. Its tasks include key management, processing cryptographic operations and maintaining the data integrity.

Each Secure Enclave comes with a unique ID (UID) during the fabrication. Other parts of the system don’t have access to UID, neither does Apple. UID is used to encrypt the Secure Enclave’s memory space and data of files stored in the file system.

The Secure Enclave is also responsible for decrypting and processing the fingerprints received from the Touch ID, verifying if the coming fingerprints match the registered fingerprints. The application processor forwards the fingerprints data to the Secure Enclave. Because the fingerprints data is encrypted with a session key between the Secure Enclave and the Touch ID, the application processor can’t read it.

2.3 Touch ID Security

Touch ID is a fingerprints sensor that can read fingerprints from the user. A user who passes the fingerprints verification can have secure access to the device, such as unlocking the iOS device, making purchases from the App Store, and making secure payment through Apple Pay (More information in the Apple Pay section).

When the user touches the home button, the capacitive steel ring on the home button detects the finger and activates the Touch ID sensor. Then Touch ID scans the fingerprints and sends a 88-by-88-pixel, 500-ppi raster scan to the Secure Enclave for authentication. The scan is vectorized for analysis and temporarily stored in encrypted memory in the Secure Enclave. After authentication, it is discarded.

3. Data Security

Besides making sure that only trusted code and apps can run on the devices, iOS also encrypts and protects user’s data locally and remotely.

3.1 Hardware security features

Because cryptographic operations are complicated and CPU consuming, they could introduce battery life problems if they are not well implemented.

Every iOS device has a dedicated AES-256 crypto engine built into the Directed Memory Access (DMA) path between the flash storage and main system memory, hence making the encryption process highly efficient.

Every device also has a unique UID and a device group ID (GID), which are 256-bit keys. They are created and stored in the application processor during fabrication, no software or hardware can access them directly. The UID is unique to each device while devices that have the common processors have the same GID. Data encrypted using the UID is tied to a particular device, hence if the memory chip is physically moved to another device, the encrypted files will not be accessible. [Apple 01]

Except the UID and GID, all other keys used for encryption are created by the iOS's random number generator (RNG) using Counter mode Deterministic Random Byte Generator (CTR-DRBG).

3.2 File Data Protection

iOS protects the file data by constructing and managing a hierarchy of keys in conjunction with hardware encryption engine. Fig.2 depicts this hierarchy and relations between the keys. When a file is created, the Data Protection system creates a 256-bit key ("per-file" key) and forwards it to the hardware AES engine, which will use the per-file key to encrypt the file.

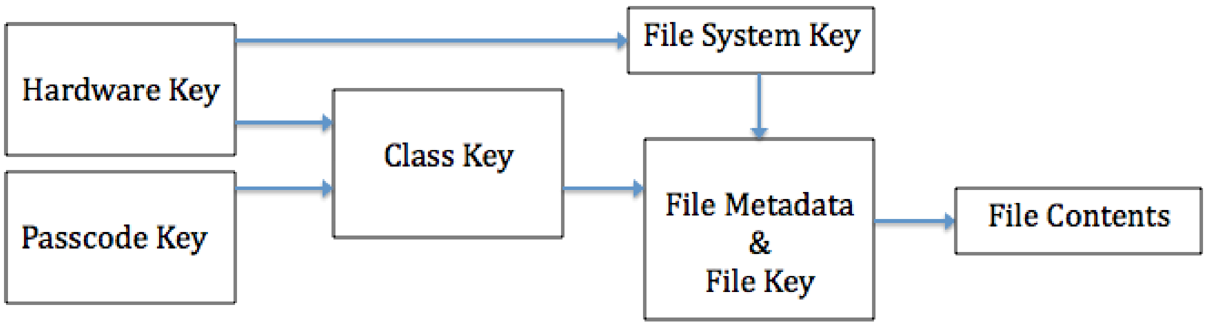


Fig.2 Hierarchy of iOS file encryption keys

As shown in Fig.2, the per-file key is wrapped with corresponding class keys. The key wrapping process is implemented using NIST AES key wrapping. The resulting wrapped per-file key is stored in the file's metadata. To open a file, iOS first decrypts the file's meta data using the file system key, having the wrapped per-file key and classes which are used to wrap the per-file key. Then iOS unwraps the per-file key using the corresponding class keys, and forwards the per-file key to the hardware AES engine. Then the AES engine decrypts the file contents read from the flash storage.

3.3 Passcodes

Passcode is an important element to iOS security. By setting up a passcode, Data Protection is automatically enabled by iOS. Most people think that passcode is just a 4 digit passcode for unlocking the iOS devices. Nevertheless, it can be set as an arbitrary length alphanumeric passcode. More importantly, the passcode is also used for generating encryption keys. Hence the stronger your passcode is, the stronger encryption keys are generated.

3.4 Data Protection Class

When a file is created on the iOS device, it is assigned a class by the app that creates it. A class is used to specify the policies on when the data is accessible. Some examples of typical classes are Complete Protection, Protected Unless Open, Protected Until First User Authentication, No Protection.

3.5 Keychain Security

iOS apps deal with a lot of sensitive data, such as passwords. In order to store these passwords securely, iOS keychain is used.

An keychain item contains metadata, such as the creation/modification timestamps, and the access group of this keychain. Besides, a keychain contains the version number, the access control list, a value indicating which protect class the item is in, a per-item key wrapped with the protection class key and a dictionary of attributes describing the item. All these contents are encrypted using AES 128 in Galois/Counter Mode (GCM).

4. App Security

Applications are one of the most important elements in iOS. As of June 2014, Apple's App Store contained more than 1.2 million iOS applications, which have collectively been downloaded more than 60 billion times. While apps bring users incredible productivity and pleasing user experience, the existence of unauthorized malware is still a big threat to iOS security.

To make sure all the apps running on iOS are not performing malicious tasks, Apple encores that all the apps must be reviewed and approved by Apple before being available on the App Store.

4.1 App review and code signing

In order for an application to be installed on iOS devices, Apple requires that all executable code be signed by Apple issued certificate. This requirement significantly reduces the threats of malware and filters out buggy apps.

In order for developers to develop and sell their apps on the App Store, developers must register and join the Apple Developer Program, obtaining a verified developer certificate. Developers must use this certificate to submit their apps to App Store, after being reviewed and approved by Apple, the apps will be available on the App Store.

4.2 Runtime process security

iOS uses a mechanism called Sandbox to prevent apps from compromising other apps installed on the devices. Apps are restricted from accessing files associated with other apps and making secure changes to the device. In addition, 3rd-party apps have limited resources when performing background tasks. iOS allows only system provided APIs to be called by 3rd-party apps in background. [Al-Qershi 02]

4.3 Data Protection in Apps

iOS Software Development Kit (SDK) provides a lot of APIs such that developers can easily write Data Protection code. Examples of these APIs are NSFileManager, CoreData, NSData, SQLite.

5. Network and Internet Services Security

There are a lot of ongoing network activities while people are using their iOS devices. To ensure that the data and user's privacy are protected during the transmission over network. iOS incorporates lots of advanced technologies and the latest standard network protocols.

5.1 SSL, TLS

iOS supports Secure Socket Layer (SSL v3) as well as Transport Layer Security (TLS v1.0, TLS v1.1, TLS v1.2). Built-in apps such as Safari, Mail use these protocols securely communicate with remote servers. iOS also provides APIs such as CFNetwork and SecureTransport so that developers could easily set up and maintain a secure SSL or TLS networking session, though the details of implementation are not open to public.

5.2 Wifi

For Wi-Fi, iOS supports WPA2 Enterprise, which uses 128-bit AES encryption. Also, iOS supports 802.1X and Remote Authentication Dial In User Service (RADIUS), the following 802.1X wireless authentication methods are supported: EAP-TLS, EAP-TTLS, EAP-FAST, EAP-SIM, PEAPv0, PEAPv1, and LEAP. In addition, a randomized Media Access Control (MAC) address is used when iOS is performing Preferred Network Offload (PNO) scans. PNO scans happen when an iOS device is not connected to a Wi-Fi network and its processor is asleep, then PNO scans decide if the device can connect to a preferred Wi-Fi network. Because the MAC address changes, it is impossible for an attacker or an observer to track an iOS device's traffic, which brings much more security.[Apple 01]

5.3 AirDrop security

AirDrop allows users to share files on their iOS or OS X devices with nearby users. When a user enables AirDrop, a 2048-bit RSA identity and its hash are created and stored on the device. When AirDrop is open, a signal is emitted through Bluetooth Low Energy such that nearby devices that also have AirDrop turned on can receive it. After the sender chooses to whom he/she wants to send, a TLS connection is created between the sender and the receiver, with iCloud identity certificates being exchanged. After the receiver accepts the files to transfer, the transmission begins.

5.4 iMessage Security

iMessage is a very popular message service provided by Apple. Users can communicate with each other through iMessage as long as one of them is using an iOS or OS X devices. Now iMessage even supports receiving SMS text messages on an OS X device. Besides text, users can also send attachments such as photos and documents. iMessage extensively uses Apple Push Notification service (APNs), which is responsible for propagating information to iOS and OS X devices in a secure way. [Apple 14]

When iMessage is turned on, an RSA 1280-bit key for encryption and an Elliptic Curve Digital Signature Algorithm (ECDSA) 256-bit key for signing are generated. The private keys for those two keys are stored in the device's keychain. The public keys are sent to Apple's directory service, which is responsible for storing user's identification, his/her associated public keys and devices' APNs addresses.

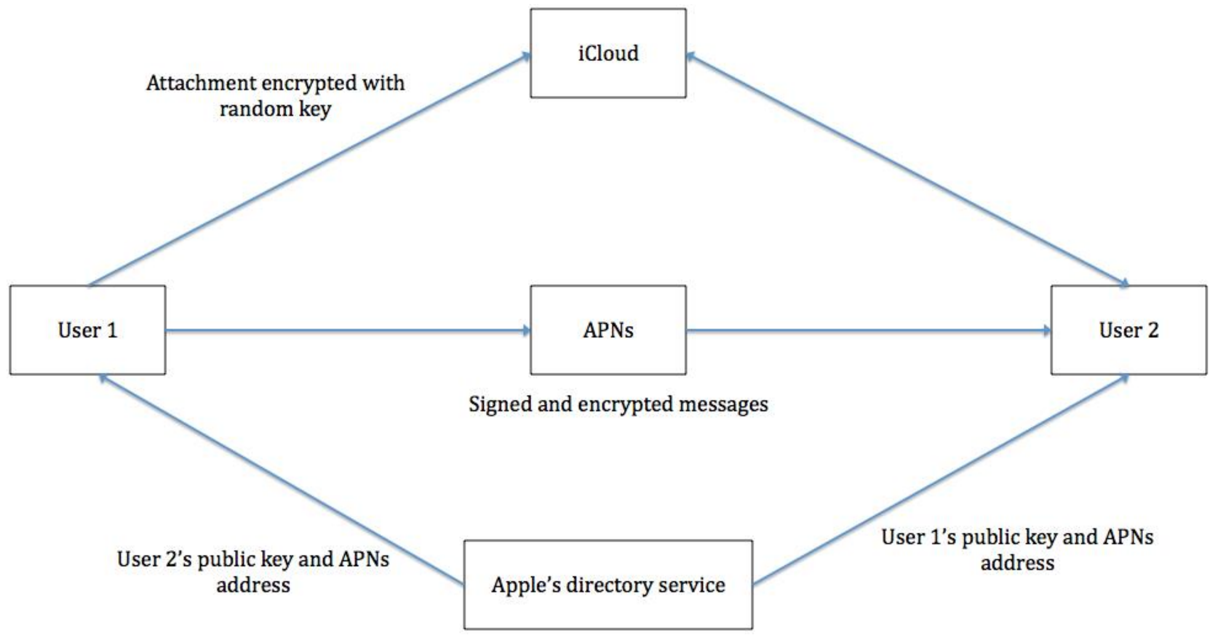

iCloud
Attachment encrypted with random key
User 1
APNs
User 2
Signed and encrypted messages
User 2's public key and APNs address
Apple's directory service
User 1's public key and APNs address

Fig.3 iMessage flowchart

As depicted in Fig.3, when the user starts a new iMessage conversation, after entering the email address/phone number or the name of the receiver, iOS will query Apple's directory service for the receiver's public keys and APNs addresses (A user can sign in iMessage on multiple devices). The message sent by the sender is encrypted by AES-128 in CTR mode for each device of the receiver, and it is signed using the sender's private key. Then the message will be pushed to the APNs with unencrypted metadata, such as the timestamp and APNs routing information. If the message is too long, for example, a large size attachment. Then instead of pushing it to the APNs, the attachment will be encrypted with a random key and uploaded to iCloud, then the receiver will fetch that attachment from iCloud.

During the transmission, the message is delivered as a copy from APNs to the receiver, once the message is successfully delivered, APNs will delete it.

5.5 FaceTime Security

Users can make video calls on iOS or OS X devices with others using FaceTime. It establishes an end-to-end connection between correspondents using Session Initiation Protocol. The contents of the communication is encrypted such that only the sender and receiver can decrypt them.

5.6 iCloud Security

iCloud is used to store contacts, photos, calendar and other documents such that they are synchronized on all of the user's iOS devices. Recently, iCloud drive is released so that users can upload any kind files to iCloud. iCloud keychain is used to synchronize user's passwords across different devices.

Files stored on iCloud are broken into blocks and encrypted using AES-128 and a key that is derived from SHA-256 hash of the block's contents. Note that the encrypted blocks of files, without user's identity information, are not stored on Apple's data center, instead they are stored on 3rd-party storages such as Amazon S3 and Windows Azure.

5.7 Continuity and Handoff

Starting from OS X Yosemite, Apple introduces Continuity and Handoff such that iPhone can be more synchronized with other iOS and OS X devices. With Continuity, a user’s Mac, iPad that share the same Wi-Fi network with his/her iPhone can also make and receive phone calls as user’s iPhone. The audio received from iPhone is transmitted to associated Mac/iPad through an encrypted end-to-end connection. The connection is established through APNs. Handoff is similar to Continuity, a user can conveniently continue working on the same task when he/she switches to another device. Similar to Continuity, two devices establish a Bluetooth Low Energy 4.0 connection through APNs. Then each generates a 256-bit AES key. Key exchange happens at the beginning of the communication. Exchanged keys will be used to encrypt and authenticate the messages sent through Bluetooth in GCM.

6. Apple Pay Security

Apple Pay to makes payments so easy that users only need to touch the Home Button to authorize and complete the transaction. Besides its easiness, Apple Pay also ensures that the transaction is performed in a secure, private way. One thing worth noticing is that Apple doesn’t collect any transaction data during this process, and Apple doesn’t know where you bought, what you bought and how much you paid.

6.1 Apple Pay Component

Secure Element: A certified chip hosts the applets that are used to manage Apple Pay. Encrypted credit/debit card information is sent from these applets.

Near Field Communication (NFC) controller: As a gateway to the Secure Element, NFC controller handles the communications between the application processor and Secure Element, and between the Secure Element and the Point-of-sale terminal.

Passbook: Users add and manage credit/debit card using Passbook. Users can also view the card information (Last 4 digits of the card number, the Device Account Number) and the privacy policy of the bank.

Secure Enclave and Touch ID: Users authorize each payment using Touch ID or his/her passcode. The Secure Enclave verifies if the fingerprints or the password is valid, if so the payment can proceed.

Apple Pay Servers: The Apple Pay Servers communicate with the devices and the payment network servers to manage the state of the credit/debit cards. [Apple 01]

6.2 Credit and debit card provisioning

When a new debit/credit card is added to Apple Pay, Apple securely sends the card information to the card’s bank, the bank will then decide whether this card can be used in Apple Pay. All the communication sessions with the bank are encrypted using SSL.

A unique Device Account Number is created for each newly added card. This Device Account Number is encrypted and stored in the Secure Element, which is isolated from iOS and Apple Pay server. The card’s bank can deactivate this Device Account Number if the user no longer wants to use it in Apple Pay. One thing worth noting is that the card’s information (card number, expiration date, etc) is not stored on the device.

6.3 How the payment is done by Apple Pay

When making a payment, the user can use Touch ID to authenticate his/her fingerprints. Once it is validated, the Secure Enclave will notify the Secure Element that the authorization is valid. Then the Secure Element will go ahead and allow the payment. The secure communication between the Secure Element and Secure Enclave is performed using a shared pair key which is created during the fabrication process. The encryption and authentication of this communication is based on AES, with cryptographic nonces on both sides to prevent replay attacks.

During the transaction, a one-time dynamic security code is generated using a key that's provisioned in the Security Element applet, along with a counter that increments for each new transaction, a random number generated by the applet and another random number generated by the server or the terminal. The payment network and the bank can verify this transaction using the dynamic security code and the Device Account Number, hence the full credit/debit card number is never sent to the merchant

7. Issues with iOS security

Because iOS is a very advanced and secure mobile operation system, and Apple has strict app review procedure, it is almost impossible for malicious-looking apps to be published on the App Store and perform malicious tasks on iOS devices. However, there still exists some threats and loopholes in iOS security.

7.1 Benign apps could become evil

A research team claims that they successfully published their malicious app (called Jekyll app) on the App Store by hiding the malicious behaviors of the app during the app review process.

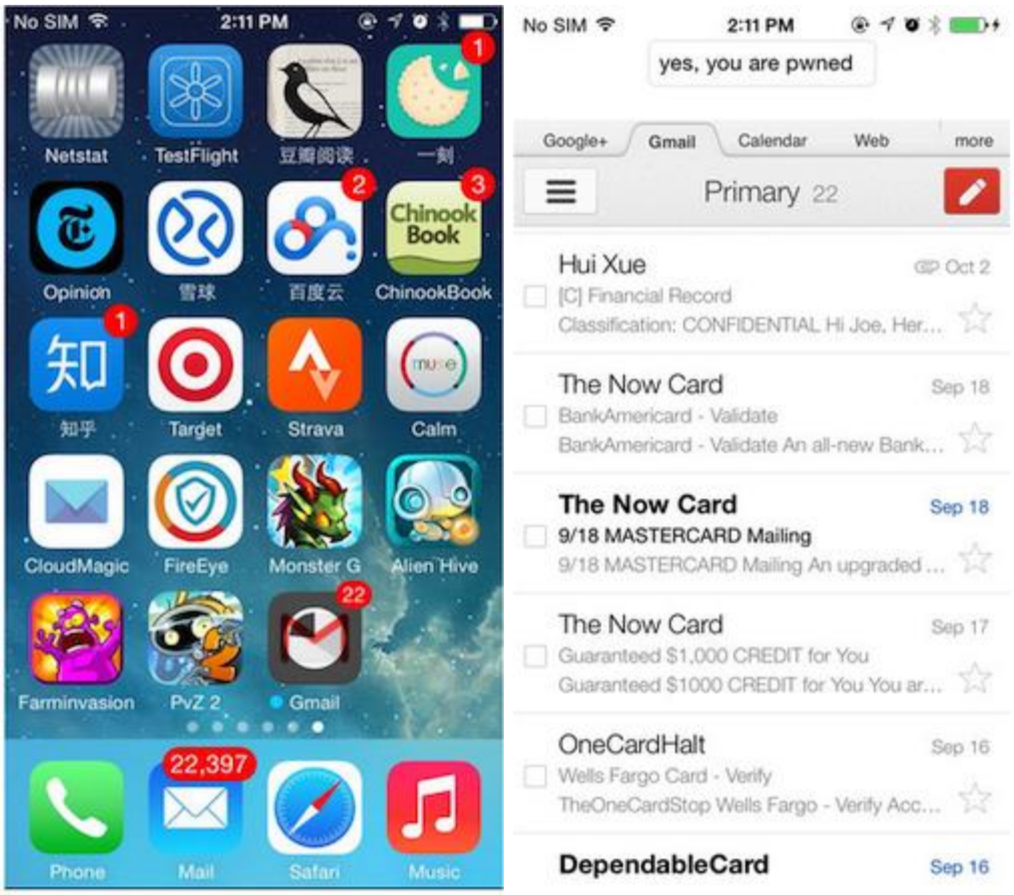
As they described in their paper, to walk around Apple's strict app review and code signing procedure, malicious attackers hide malicious behaviours in the app such that it looks legitimate and benign during the review process. After the user downloads and runs the app, the attacker can remotely activate the malicious functionalities (i.e., backdoor) in the app. Specifically, as presented in the paper, Jekyll app can post tweets, send emails and SMS texts without the user's knowledge.

7.2 Unauthorized origin-crossing threats

In computing, the same-origin policy is an important concept in the web application security model. The policy permits scripts running on pages originating from the same site – a combination of scheme, hostname, and port number – to access each other's DOM with no specific restrictions, but prevents access to DOM on different sites. In contrast, the cross-origin is a mechanism that allows many resources (e.g., fonts, JavaScript, etc.) on a web page to be requested from another domain outside the domain from which the resource originated.

Unlike web browsers that have the same-origin security policy, mobile operating systems don't have such policy, as a result, users' sensitive data could be exploited by malicious web origin. Researches found cross-origin issues in popular SDKs and high-profile apps such as Facebook and Dropbox, which can be exploited to access their users' authentication credentials and other confidential information. [Wang 04]

7.3 Masque Attack



yes, you are pwned

Hui Xue — Oct 2
[C] Financial Record
Classification: CONFIDENTIAL Hi Joe, Her...

The Now Card — Sep 18
BankAmericard - Validate
BankAmericard - Validate An all-new Bank...

The Now Card — **Sep 18**
9/18 MASTERCARD Mailing
9/18 MASTERCARD Mailing An upgraded ...

The Now Card — Sep 17
Guaranteed $1,000 CREDIT for You
Guaranteed $1000 CREDIT for You You ar...

OneCardHalt — Sep 16
Wells Fargo Card - Verify
TheOneCardStop Wells Fargo - Verify Acc...

DependableCard — **Sep 16**

Fig.4 Screenshot after Masque Attack

A mobile security research team called FireEye recently claim that attackers can fool users to install malicious iOS apps which disguises itself as other commonly used apps, such as Gmail app or even your bank app. As long as the two apps use the same bundle identifier, the genuine app can be replaced by the malicious app in a way that looks exactly the same as the process of updating an existing app, as shown in Fig.4. This kind of masquerade increases the chance of tricking careless users.

This Masque attack could result in severe consequences. If the malicious app replaces your Gmail or bank app, after you run the app, malicious code will be executed and send your login credentials to the attacker. Besides, the malicious code can access the local files of the replaced app, resulting in more private data being stolen.

7.4 iOS jailbreaking

iOS jailbreaking is the process of removing limitations on iOS, Apple's operating system on devices running it through the use of software and hardware exploits. The reason that many people want to jailbreak their iOS devices is because it gives users root access to iOS, making users have more privileges on customizing their iOS devices. Such privileges include installing applications or extensions that are not provided by Apple’s App Store, and personalizing the user interface of iOS.

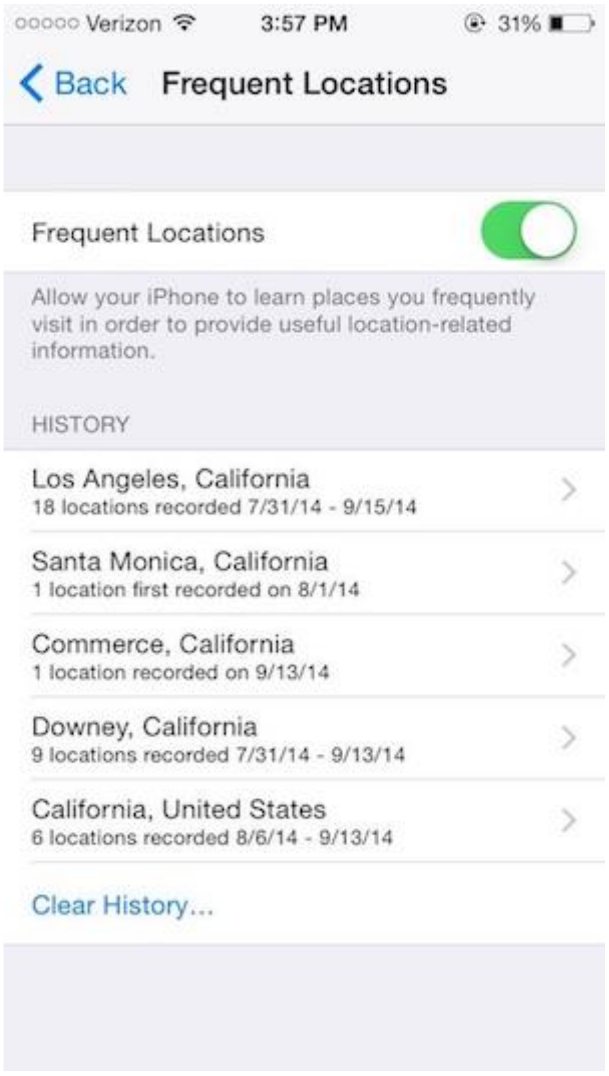
The biggest problem of jailbreaking is that it puts users into huge security risks since it disables the Sandbox feature of iOS. Sandbox is an important iOS security feature during runtime process, it separates the applications installed on the device such that apps are restricted from accessing files associated with other files during runtime. If a user installs a malicious app on the jailbroken iOS device, the malicious code will have access to address book, photos, location data and other private data without telling the user. Besides the security risks brought by jailbreaking, once a user jailbreaks his/her iOS devices, the warranty (AppleCare) provided by Apple will be void immediately. Meanwhile, as Apple has constantly been incorporating new features into the latest version of iOS, the benefits brought by jailbreaking are becoming less and less. [O'Donnell 12]][Emspak 13]

7.5 Privacy issues

Nowadays, companies are desperately trying to collect their users' data in order for them to put appropriate advertisements in their app. However, users are not aware of their personal information or even sensitive data being secretly transmitted. In the recent release of iOS 8 by Apple, iOS enables some private settings by default while users might not know them at all. Specifically, here are some notable privacy concerns.

iOS keeps track your moves all the time

Since iOS 7, iOS starts tracking your frequent locations by default, such as your home and workplace, as shown in Fig.5. While it might be somewhat useful for traffic alert, most people don't want Apple to know their frequent moves.



Frequent Locations

Allow your iPhone to learn places you frequently visit in order to provide useful location-related information.

HISTORY

Los Angeles, California
18 locations recorded 7/31/14 - 9/15/14

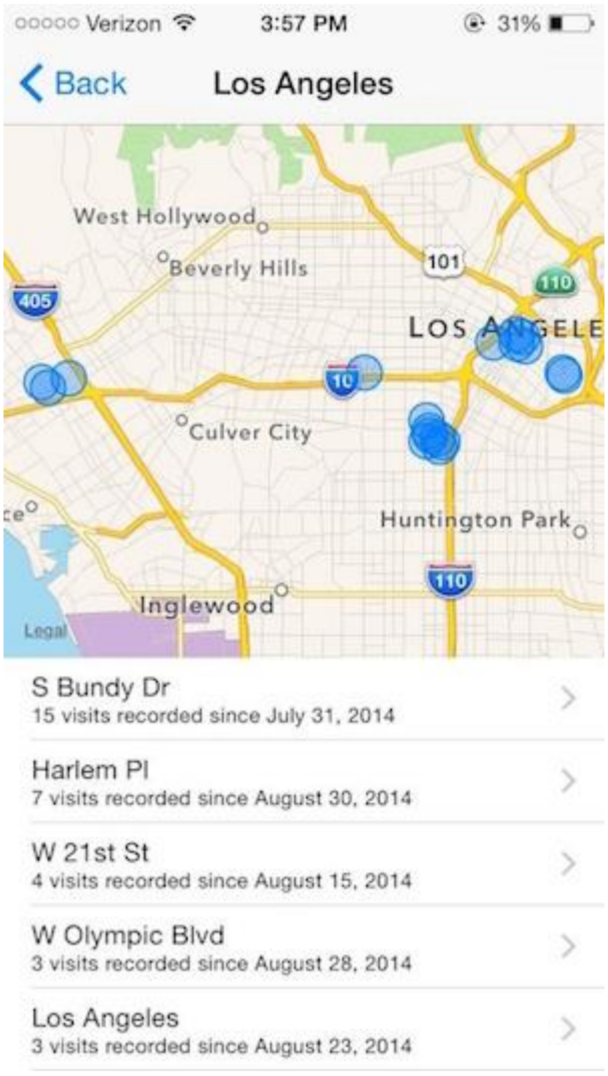
Santa Monica, California
1 location first recorded on 8/1/14

Commerce, California
1 location recorded on 9/13/14

Downey, California
9 locations recorded 7/31/14 - 9/13/14

California, United States
6 locations recorded 8/6/14 - 9/13/14

Clear History...



S Bundy Dr
15 visits recorded since July 31, 2014

Harlem Pl
7 visits recorded since August 30, 2014

W 21st St
4 visits recorded since August 15, 2014

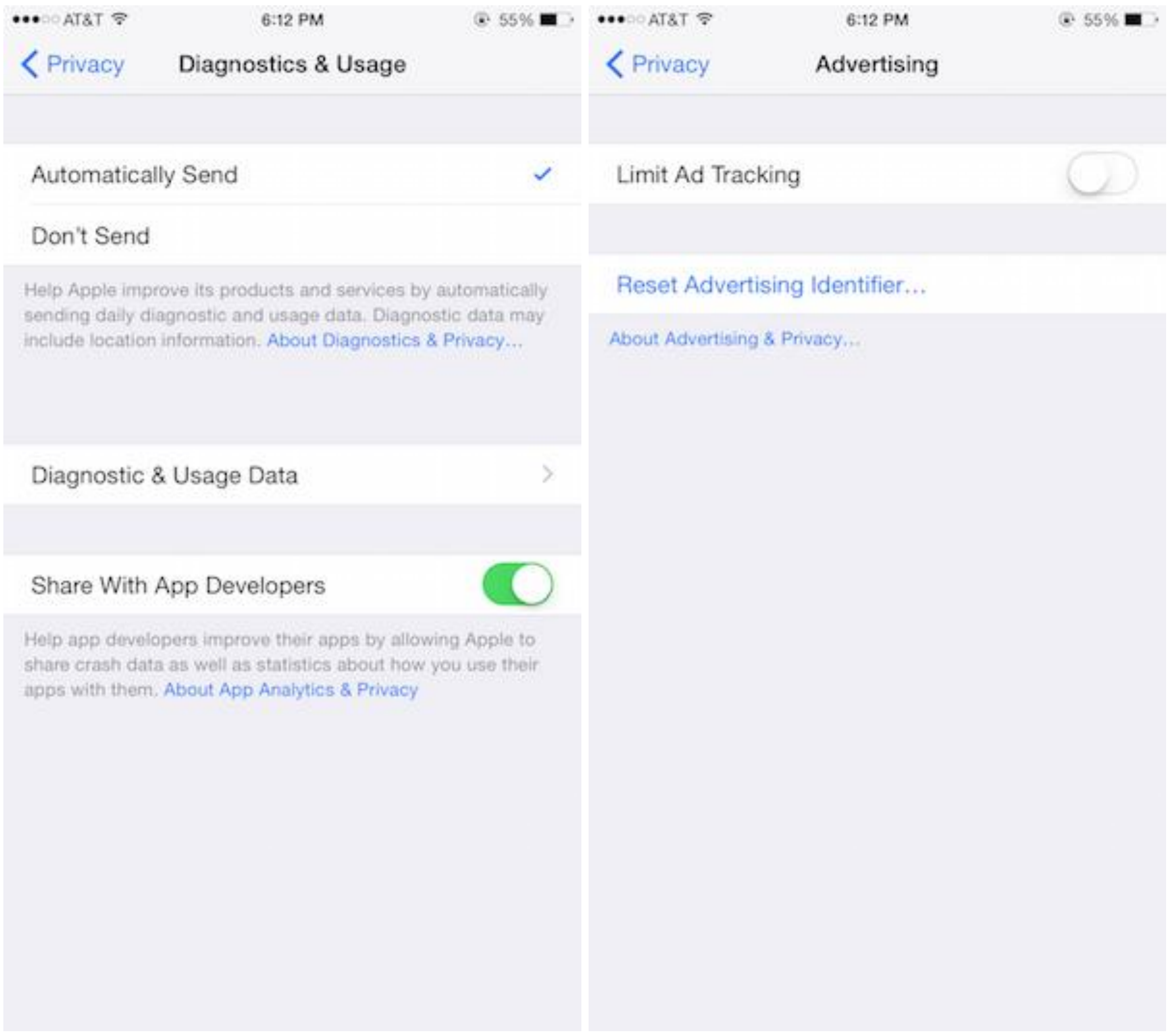
W Olympic Blvd
3 visits recorded since August 28, 2014

Los Angeles
3 visits recorded since August 23, 2014

Fig.5 Screenshots of system location privacy settings

Information sent to advertisers and developers

iOS automatically sends your diagnostics and usage data to advertisers and developers by default, as shown in Fig.6.



●●●○○ AT&T 6:12 PM 55%

Privacy — Diagnostics & Usage

Automatically Send ✓

Don't Send

Help Apple improve its products and services by automatically sending daily diagnostic and usage data. Diagnostic data may include location information. About Diagnostics & Privacy...

Diagnostic & Usage Data

Share With App Developers: on

Help app developers improve their apps by allowing Apple to share crash data as well as statistics about how you use their apps with them. About App Analytics & Privacy

●●●○○ AT&T 6:12 PM 55%

Privacy — Advertising

Limit Ad Tracking: off

Reset Advertising Identifier...

About Advertising & Privacy...

Fig.6 Screenshots of diagnostics and usage data settings

Reversing Android Application Package

Reverse engineering refers to the process of taking something apart to see how it works, whether it's a physical object such as a lock or in this case, a mobile application. Decompiling is a form of reverse engineering in which a mobile app is analyzed by looking at its source code. A decompiler program examines the binary and translates its contents from a low-level abstraction to a higher-level abstraction.

An Android binary is called an APK, which stands for Android Package Kit. The APK contains application data in the form of zipped Dalvik Executable (.dex) files. DEX files consist of the following components:

- File Header
- String Table

- Class List
- Field Table
- Method Table
- Class Definition Table
- Field List
- Method List
- Code Header
- Local Variable List

A decompiler is a tool that takes the contents of the APK and attempts to show the original code that was used to build the different functionalities of the app. Although this process is consistent, in some special cases the decompiler might fail to reverse a small part of the application.

Analyze the “InsecureBankv2.apk”, an open-source Android application that purposefully contains many vulnerabilities.

Android Application Signing

- Application signing allows developers to identify the author of the application and to update their application without creating complicated interfaces and permissions. Every application that is run on the Android platform must be signed by the developer. Applications that attempt to install without being signed will be rejected by either Google Play or the package installer on the Android device.
- On Google Play, application signing bridges the trust Google has with the developer and the trust the developer has with their application. Developers know their application is provided, unmodified, to the Android device; and developers can be held accountable for behavior of their application.
- On Android, application signing is the first step to placing an application in its Application Sandbox. The signed application certificate defines which user ID is associated with which application; different applications run under different user IDs. Application signing ensures that one application cannot access any other application except through well-defined IPC.
- When an application (APK file) is installed onto an Android device, the Package Manager verifies that the APK has been properly signed with the certificate included in that APK. If the certificate (or, more accurately, the public key in the certificate) matches the key used to sign any other APK on the device, the new APK has the option to specify in the manifest that it will share a UID with the other similarly-signed APKs.
- Applications can be signed by a third-party (OEM, operator, alternative market) or self-signed. Android provides code signing using self-signed certificates that developers can generate without external assistance or permission. Applications do not have to be signed by a central authority. Android currently does not perform CA verification for application certificates.
- Applications are also able to declare security permissions at the Signature protection level, restricting access only to applications signed with the same key while maintaining distinct UIDs and Application Sandboxes. A closer relationship with a shared Application Sandbox is allowed

using the shared UID feature where two or more applications signed with same developer key can declare a shared UID in their manifest.

Note: As of Android 13, shared UIDs are deprecated. Users of Android 13 or higher should put the line **android:sharedUserMaxSdkVersion="32"** in their manifest. This entry prevents new users from getting a shared UID.

APK signing schemes

Android supports three application signing schemes:

- v1 scheme: based on JAR signing
- v2 scheme: APK Signature Scheme v2, which was introduced in Android 7.0.
- v3 scheme: APK Signature Scheme v3, which was introduced in Android 9.

For maximum compatibility, sign applications with all schemes, first with v1, then v2, and then v3. Android 7.0+ and newer devices install apps signed with v2+ schemes more quickly than those signed only with v1 scheme. Older Android platforms ignore v2+ signatures and thus need apps to contain v1 signatures.

JAR signing (v1 scheme)

APK signing has been a part of Android from the beginning. It is based on signed JAR.

v1 signatures do not protect some parts of the APK, such as ZIP metadata. The APK verifier needs to process lots of untrusted (not yet verified) data structures and then discard data not covered by the signatures. This offers a sizeable attack surface. Moreover, the APK verifier must uncompress all compressed entries, consuming more time and memory. To address these issues, Android 7.0 introduced APK Signature Scheme v2.

APK Signature Scheme v2 & v3 (v2+ scheme)

Devices running Android 7.0 and later support APK signature scheme v2 (v2 scheme) and later. (v2 scheme was updated to v3 in Android 9 to include additional information in the signing block, but otherwise works the same.) The contents of the APK are hashed and signed, then the resulting APK Signing Block is inserted into the APK.

During validation, v2+ scheme treats the APK file as a blob and performs signature checking across the entire file. Any modification to the APK, including ZIP metadata modifications, invalidates the APK signature. This form of APK verification is substantially faster and enables detection of more classes of unauthorized modifications.

The new format is backwards compatible, so APKs signed with the new signature format can be installed on older Android devices (which simply ignore the extra data added to the APK), as long as these APKs are also v1-signed.

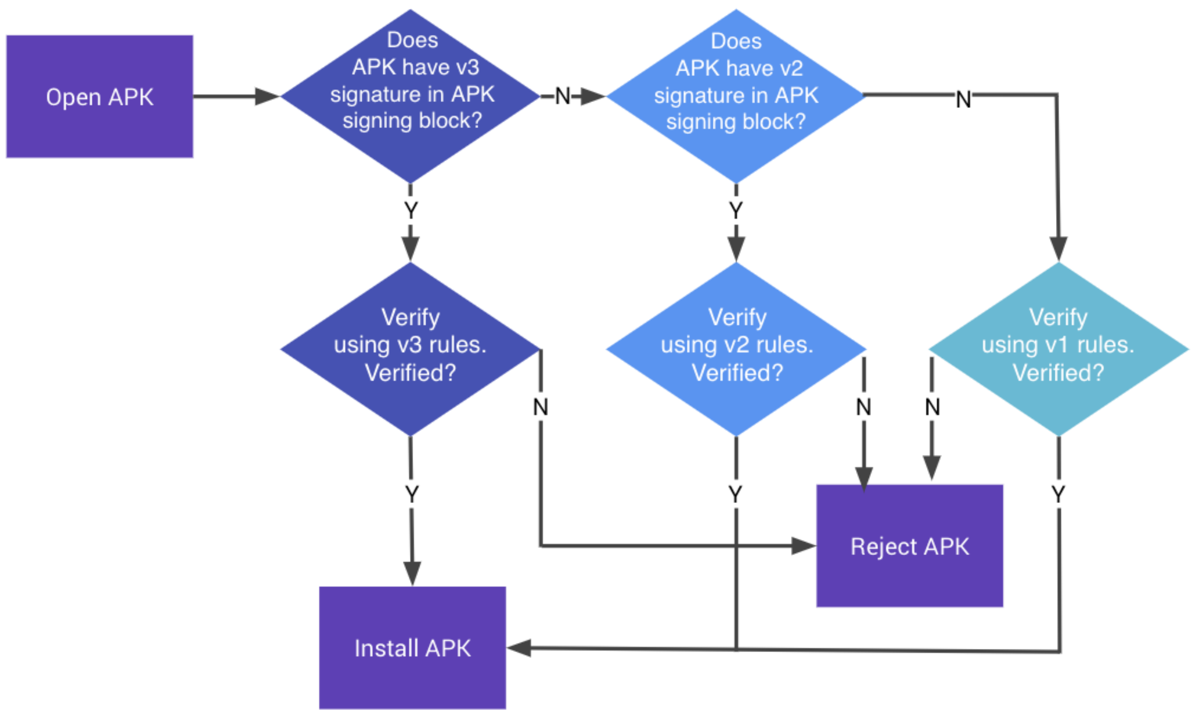

Open APK
Does APK have v3 signature in APK signing block?
Does APK have v2 signature in APK signing block?
Verify using v3 rules. Verified?
Verify using v2 rules. Verified?
Verify using v1 rules. Verified?
Reject APK
Install APK
Y
N

Figure 1. APK signature verification process

Whole-file hash of the APK is verified against the v2+ signature stored in the APK Signing Block. The hash covers everything except the APK Signing Block, which contains the v2+ signature. Any modification to the APK outside of the APK Signing Block invalidates the APK's v2+ signature. APKs with stripped v2+ signature are rejected as well, because their v1 signature specifies that the APK was v2-signed, which makes Android 7.0 and newer refuse to verify APKs using their v1 signatures.

Android signature verification

Signature verification is a security feature on Android that helps protect the data of the applications installed from getting corrupted by the lower versions of the app, or other apps with the same names but different signatures.

By default, the Android OS requires **all** applications to be signed in order to be installed. In very basic terms, this means that the application signature is used to identify the author of an application (*i.e.* verify its legitimacy), as well as establish trust relationships between applications with the same signature. With the former, you are assured (to a reasonable degree) that an application with a valid signature comes from the expected developers. And through the latter, applications signed with the same private key may run in the same process and share private data.

Then when you install an application update, the Android OS checks this signature to make sure that: A) the APK has not been tampered in the time since it was signed, and B) the application's certificate matches that of the currently installed version.

Application Manifest File in Android

Every project in Android includes a Manifest XML file, which is **AndroidManifest.xml**, located in the root directory of its project hierarchy. The manifest file is an important part of our app because it defines the structure and metadata of our application, its components, and its requirements. This file includes nodes for each of the **Activities**, **Services**, **Content Providers**, and **Broadcast Receivers** that make the application, and using **Intent Filters** and Permissions determines how they coordinate with each other and other applications.

The manifest file also specifies the application metadata, which includes its icon, version number, themes, etc., and additional top-level nodes can specify any required permissions, unit tests, and define hardware, screen, or platform requirements. The manifest comprises a root manifest tag with a package attribute set to the project's package. It should also include an xmls:android attribute that will supply several system attributes used within the file. We use the versionCode attribute is used to define the current application version in the form of an integer that increments itself with the iteration of the version due to update. Also, the versionName attribute is used to specify a public version that will be displayed to the users.

We can also specify whether our app should install on an SD card of the internal memory using the installLocation attribute. A typical manifest file looks as:

- XML

```
?xml version="1.0" encoding="utf-8"?>
manifest xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.com/tools"
 package="com.example.geeksforgeeks"
 android:versionCode="1"
 android:versionName="1.0"
 android:installLocation="preferExternal">
     <uses-sdk
    android:minSdkVersion="18"
    android:targetSdkVersion="27" />
 <application
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.MyApplication"
    tools:targetApi="31">
    <activity
        android:name=".MainActivity"
        android:exported="true">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
             <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
 </application>
</manifest>
```

A manifest file includes the nodes that define the application components, security settings, test classes, and requirements that make up the application. Some of the manifest sub-node tags that are mainly used are:

1. manifest

The main component of the AndroidManifest.xml file is known as manifest. Additionally, the package field describes the activity class's package name. It must contain an <application> element with the xmlns:android and package attribute specified.

- XML

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.geeksforgeeks">

    <!-- manifest nodes -->

    <applicataion>
```

</**applicataion**>

</**manifest**>

2. uses-sdk

It is used to define a minimum and maximum SDK version by means of an API Level integer that must be available on a device so that our application functions properly, and the target SDK for which it has been designed using a combination of minSdkVersion, maxSdkVersion, and targetSdkVersion attributes, respectively. It is contained within the <manifest> element.

- XML

uses-sdk

android:minSdkVersion="18"

android:targetSdkVersion="27" />

3. uses-permission

It outlines a system permission that must be granted by the user for the app to function properly and is contained within the <manifest> element. When an application is installed (on Android 5.1 and lower devices or Android 6.0 and higher), the user must grant the application permissions.

- XML

uses-permission

android:name="android.permission.CAMERA"

android:maxSdkVersion="18" />

4. application

A manifest can contain only one application node. It uses attributes to specify the metadata for your application (including its title, icon, and theme). During development, we should include a debuggable attribute set to true to enable debugging, then be sure to disable it for your release builds. The application node also acts as a container for the Activity, Service, Content Provider, and Broadcast Receiver nodes that specify the application components. The name of our custom application class can be specified using the android:name attribute.

- XML

application

android:name=".GeeksForGeeks"

android:allowBackup="true"

```
 android:dataExtractionRules="@xml/data_extraction_rules"
 android:fullBackupContent="@xml/backup_rules"
 android:icon="@drawable/gfgIcon"
 android:label="@string/app_name"
 android:roundIcon="@mipmap/ic_launcher_round"
 android:supportsRtl="true"
 android:theme="@android:style/Theme.Light"
 android:debuggable="true"
 tools:targetApi="31">
 <!-- application nodes -->
/application>
```

5. uses-library

It defines a shared library against which the application must be linked. This element instructs the system to add the library's code to the package's class loader. It is contained within the <application> element.

- XML

```
uses-library
 android:name="android.test.runner"
 android:required="true" />
```

6. activity

The Activity sub-element of an application refers to an activity that needs to be specified in the AndroidManifest.xml file. It has various characteristics, like label, name, theme, launchMode, and others. In the manifest file, all elements must be represented by <activity>. Any activity that is not declared there won't run and won't be visible to the system. It is contained within the <application> element.

- XML

```
activity
 android:name=".MainActivity"
 android:exported="true">
```

/**activity**>

7. intent-filter

It is the sub-element of activity that specifies the type of intent to which the activity, service, or broadcast receiver can send a response. It allows the component to receive intents of a certain type while filtering out those that are not useful for the component. The intent filter must contain at least one <action> element.

- XML

intent-filter>

<**action** android:name="android.intent.action.MAIN" />

<**category** android:name="android.intent.category.LAUNCHER" />

/**intent-filter**>

8. action

It adds an action for the intent-filter. It is contained within the <intent-filter> element.

- XML

action android:name="android.intent.action.MAIN" />

9. category

It adds a category name to an intent-filter. It is contained within the <intent-filter> element.

- XML

category android:name="android.intent.category.LAUNCHER" />

10. uses-configuration

The uses-configuration components are used to specify the combination of input mechanisms that are supported by our application. It is useful for games that require particular input controls.

- XML

uses-configuration

android:reqTouchScreen=”finger”

android:reqNavigation=”trackball”

android:reqHardKeyboard=”true”

android:reqKeyboardType=”qwerty”/>

uses-configuration

android:reqTouchScreen=”finger”

android:reqNavigation=”trackball”

android:reqHardKeyboard=”true”

android:reqKeyboardType=”twelvekey”/>

11. uses-features

It is used to specify which hardware features your application requires. This will prevent our application from being installed on a device that does not include a required piece of hardware such as NFC hardware, as follows:

- XML

uses-feature android:name=”android.hardware.nfc” />

12. permission

It is used to create permissions to restrict access to shared application components. We can also use the existing platform permissions for this purpose or define your own permissions in the manifest.

Types of permission include:

CALENDAR

READ_CALENDAR

WRITE_CALENDAR

CAMERA

CAMERA

CONTACTS

READ_CONTACTS
WRITE_CONTACTS
GET_ACCOUNTS

LOCATION

ACCESS_FINE_LOCATION
ACCESS_COARSE_LOCATION
MICROPHONE

RECORD_AUDIO
PHONE

READ_PHONE_STATE
READ_PHONE_NUMBERS
CALL_PHONE
ANSWER_PHONE_CALLS (must request at runtime)
READ_CALL_LOG
WRITE_CALL_LOG
ADD_VOICEMAIL
USE_SIP
PROCESS_OUTGOING_CALLS
ANSWER_PHONE_CALLS
SENSORS

BODY_SENSORS
SMS

SEND_SMS
RECEIVE_SMS
READ_SMS
RECEIVE_WAP_PUSH
RECEIVE_MMS

STORAGE

READ_EXTERNAL_STORAGE
WRITE_EXTERNAL_STORAGE