brightsight®

the number one
security lab
in the world

# Android

## Introduction and Security Model

Shiqi Li / Brightsight

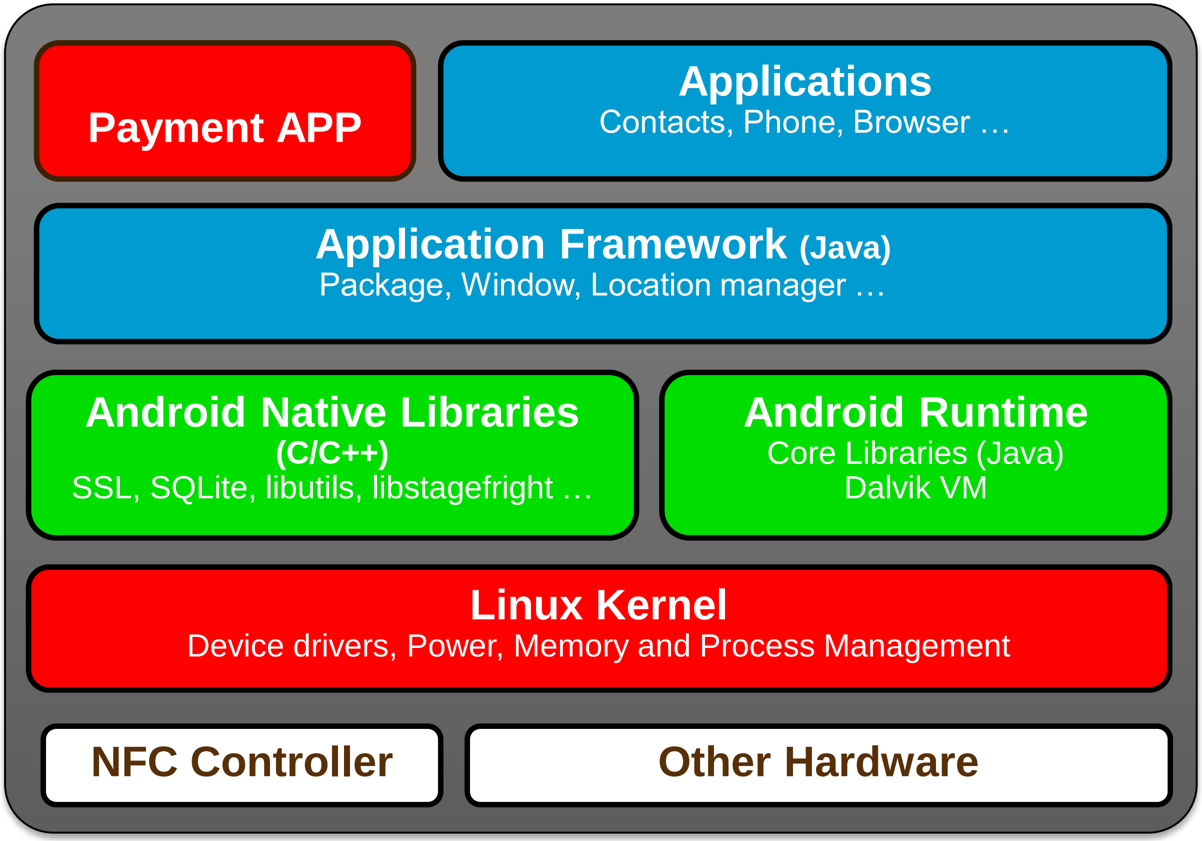
# Outline

- ❑ **Android Introduction**
- ❑ Android Security Model

# OS Internals

- Android is a software stack for mobile devices that includes an operating system, middleware and key applications.
- Android OS is a watered version of Linux. Optimized for running in mobile and embedded devices.
- Android is Linux-based although apps are usually developed against APIs that abstract anything Linux-specific.
- The OS mainly provides a platform to run instances of the DVM (Dalvik Virtual Machine).

# Android Architecture


Payment APP
Applications
Contacts, Phone, Browser …
Application Framework (Java)
Package, Window, Location manager …
Android Native Libraries
(C/C++)
SSL, SQLite, libutils, libstagefright …
Android Runtime
Core Libraries (Java)
Dalvik VM
Linux Kernel
Device drivers, Power, Memory and Process Management
NFC Controller
Other Hardware

# Application Framework

**Application Framework (Java)**
Package, Window, Location manager …

## Enabling and simplifying the reuse of components

The blocks that applications directly interact with.

- **Activity Manager**: Manages the activity life cycle of applications
- **Content Providers:** Manage the data sharing between applications
- **Telephony Manager:** Manages all voice calls. We use telephony manager if we want to access voice calls in our application.
- **Location Manager:** Location management, using GPS or cell tower
- **Resource Manager:** Manage the various types of resources we use in our Application

# Android Runtime (Core Libraries)

- Core Libraries
  - Providing most of the functionality available in the core libraries of the Java language
    - APIs
    - Data Structures
    - Utilities
    - File Access
    - Network Access
    - Graphics
    - Etc.

# Android Linux Kernel I

**Linux Kernel**
Device drivers, Power, Memory and Process Management

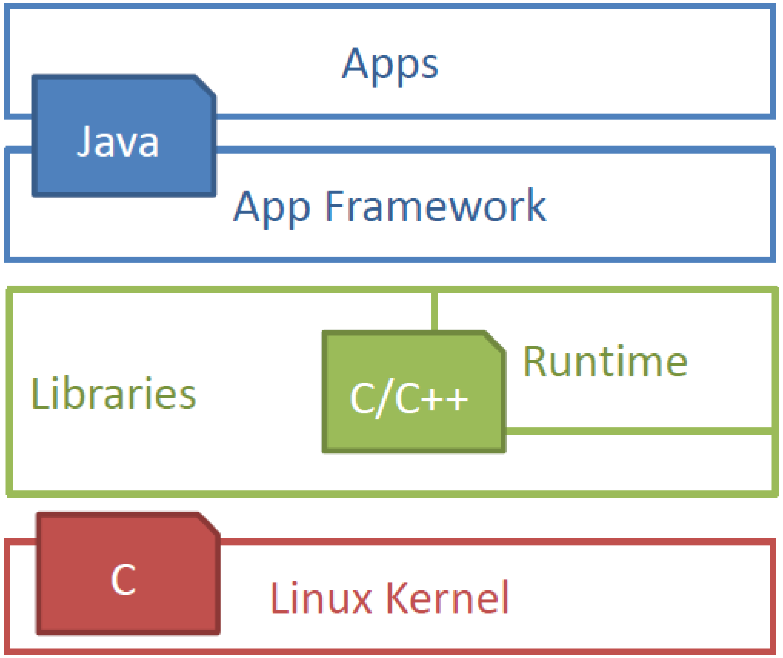
- Android relies on Linux Kernel for core system services
  - Memory and Process Management
  - Network Stack
  - Driver Model
  - Security
  - Providing an abstraction layer between the H/W and the rest of the S/W stack

# Android Linux Kernel II (OAP)

| Android Version | API Level | Linux Kernel |
|---|---|---|
| 1.5 Cupcake | 3 | 2.6.27 |
| 1.6 Donut | 4 | 2.6.29 |
| 2.0/1 Eclair | 5-7 | 2.6.29 |
| 2.2.x Froyo | 8 | 2.6.32 |
| 2.3.x Gingerbread | 9,10 | 2.6.35 |
| 3.x.x Honeycomb | 11-13 | 2.6.36 |
| 4.0.x Ice Cream Sand. | 14,15 | 3.0.1 |
| 4.1.x Jelly Bean | 16-18 | 3.4.0 |
| 4.4 Kit Kat | 19,20 | 3.10 |
| 5.x Lollipop | 21,22 | 3.16.1 |
| 6.0 Marshmallow | 23 | 3.18.10 |

# Developer’s Perspective

- Android apps are mostly developed in Java using the Android API.
- It’s possible to develop natively in C/C++ (but is not commonly done).



# Android Applications

| | |
|---|---|
| **AndroidManifest.xml**<br>Declares: app components , minimum API Level, needed API libraries, user permissions | **Activities** An activity represents a single screen with a user interface. |
| | **Content Providers** A content provider manages a shared set of application data. ***Through the content provider, other applications can query or even modify the data.*** |
| | **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. |
| | **Broadcast Receivers** A broadcast receiver is a component ***that responds to system-wide broadcast announcements***. |

# Android Application Package File (APK)

APK is the file format used to distribute and install apps and middleware onto Android OS

- APK files are ZIP archives based on JAR (Java ARchive format)
- Bundle together compiled Android classes (.dex), metadata (in META-INF/ directory) and resources the code uses.

# Android Important Partitions

/data partition contains the user data

| location | description |
|---|---|
| /data/app | Installed applications APK files |
| /data/data | Installed Application files |
| /data/dalvik-cache | Optimized DEX files |
| /data/backup | backups |

/system is a read only partition containing system default apps and data

| location | description |
|---|---|
| /system/app | Preinstalled  applications |
| /system/priv-apps | Applications running with system privileges. |

# Android Debug Bridge (ADB)

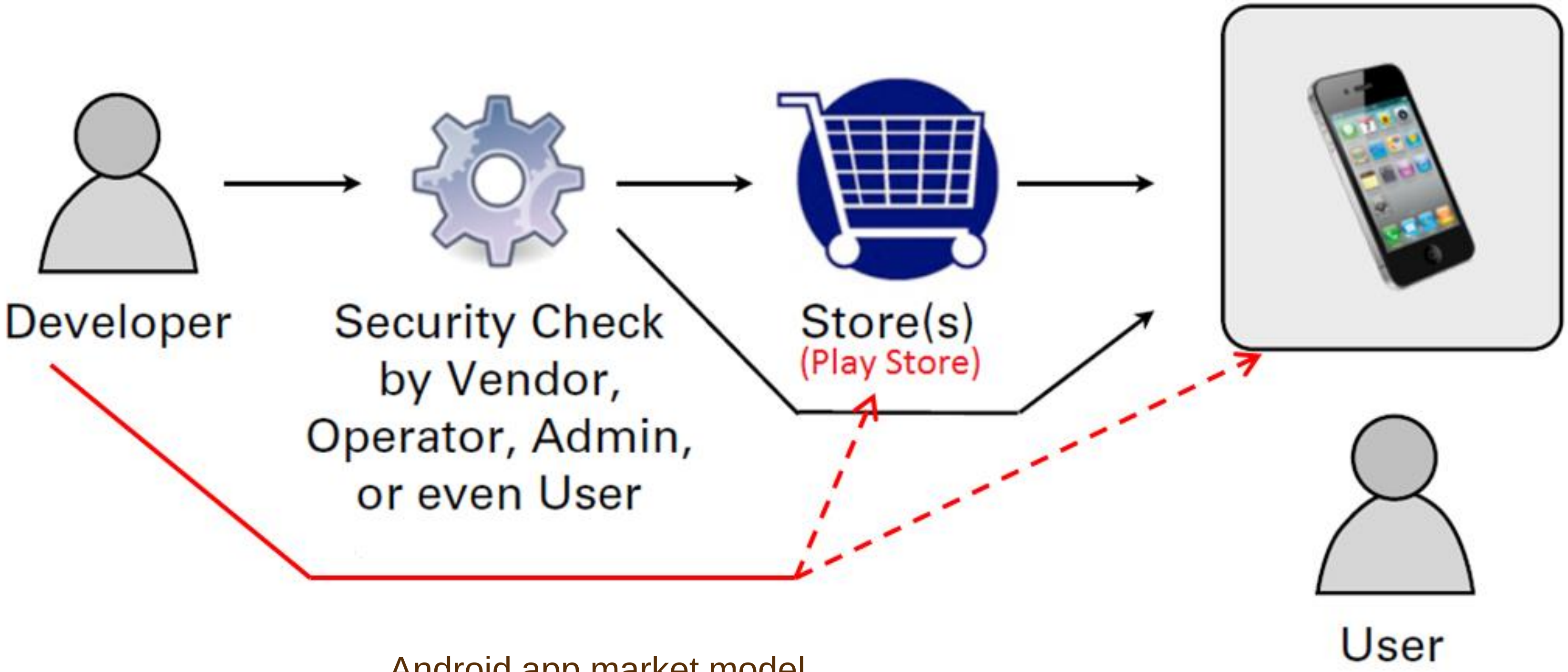
ADB is available as part of Android SDK

Disabled by default unless the developer options are active (tap 5 times on the release build info).

Makes use of a TCP/IP connection to the Android Device

- Used for a wide variety of developer tasks
  - Read from the log file
  - Show what android devices are available
  - Install android applications (.apk files)
- In the ‘platform-tools’ directory of the main android sdk directory
  - Recommend putting this directory and the ‘tools’ directory on the system path

# App market model



Android app market model

## Outline

- ❑ Android Introduction
- ❑ **Android Security Model**

# Android Security Mechanisms

- **Security Features inherited from Linux**
  - Users
  - File Access
  - SE LINUX
- **Android Specific**
  - Application Sandboxing
  - Secure inter-process communication
  - Application signing
  - Application-defined and user-granted permission
- **Environmental**
  - Memory Management Unit

# Security Inherited from LINUX

- Linux kernel is the foundation of the Android platform
- Linux is a multiuser operating system, with the main security objective to mutually isolate different users

# USERS

Each user in a Linux system is assigned a unique user ID (UID) and a group ID (GID) when they are created

Each user UID’s resource is assigned the same UID

- process, file, directory, etc.

On each access by a user or process to a given resource, the Linux kernel enforces the access control policy based on the access rights and the requestor’s UID/GUID

- The Linux kernel acts as a reference monitor

# File access

3 types of users (subjects)

- u – user who owns a file
- g – group user (all the members of the group g)
- o – all other users

3 types of permissions (access rights)

- r – read file or directory
- w – write to file or directory
- x – execute file or search directory

Given a file (object), each of the 3 access rights can be set for any of 3 the types of users by the file owner (u)

Almost everything in Linux is viewed as a file

# POSIX Permissions

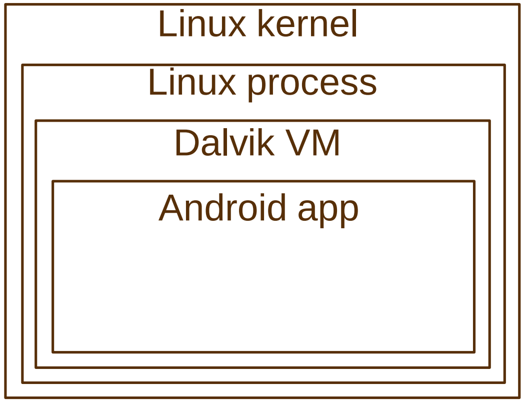
```
/adrian$ ls -l -h
total 468K
drwxr-xr-x    3 adrian grop 4.0K 2011-02-03 16:24 bkup
-rw-------    1 adrian grop  542 2007-05-22 10:26 Drafts
-rw-r--r--    1 adrian grop    0 2010-12-06 10:32 finger
drwx------    3 adrian grop 4.0K 2012-12-04 21:37 mail
drwx------ 1916 adrian grop 228K 2011-11-05 22:35 Maildir
drwxr-xr-x    2 adrian grop 4.0K 2006-08-01 15:26 MailFolders
```

# Android Sandboxing

- **Sandboxing at the Linux kernel level**
- **Each app gets a unique UID when forked from zygote and run in a separate process**
  - **Can access own files only (unless root)**
- **Memory corruption can only happen in one sandbox**

On Android each app is effectivelly a different user

- Some exceptions are possible (e.g., android:sharedUserId)



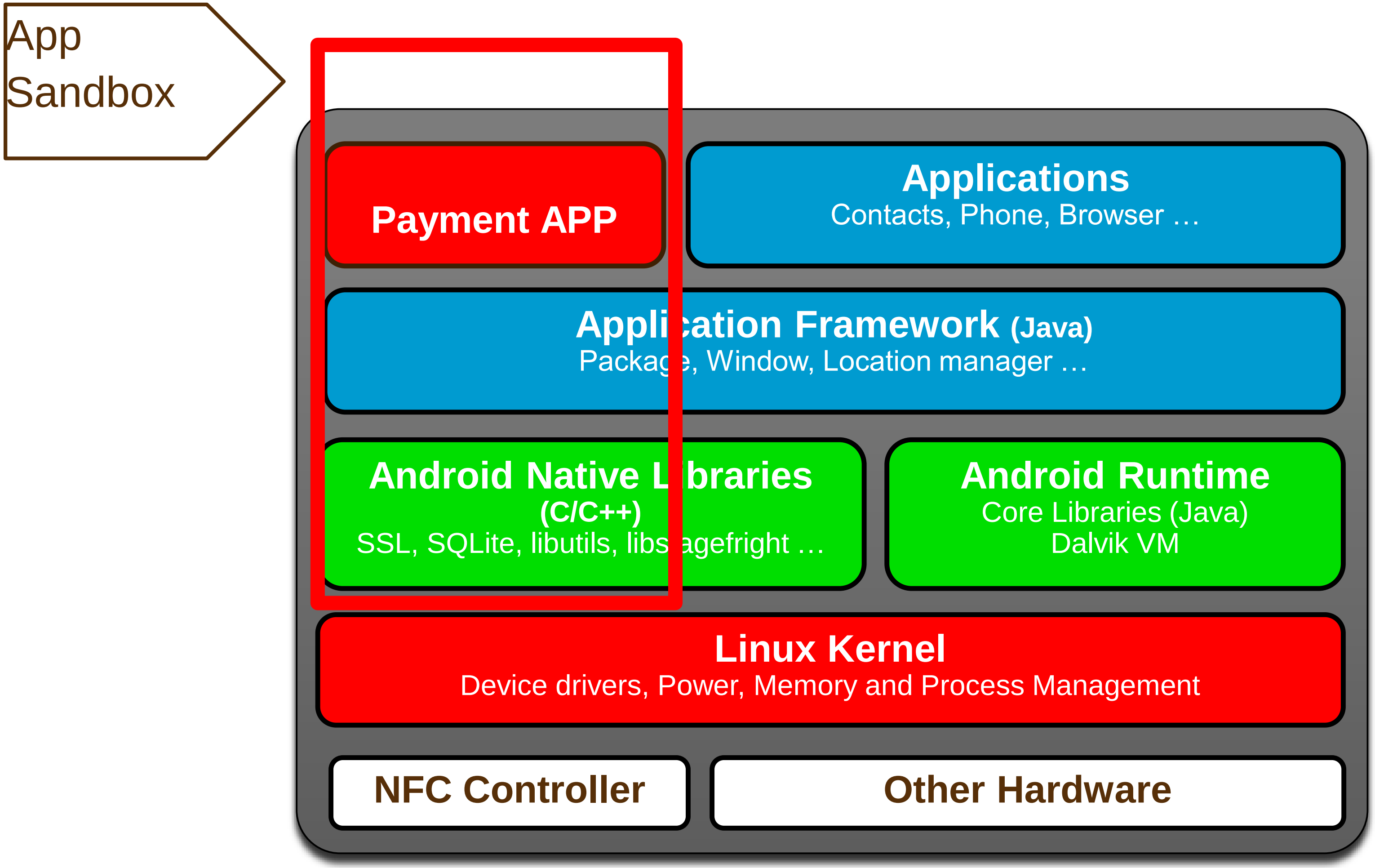
# Android Sandboxing

User IDs (UID) are assigned when an Android package is installed by the package manager.

Note /data/system/package.xml file contains info about every installed application

```
<package name=“com.example.hello” codePath=“/data/app/Hello.apk” …
         userId="10051">
      <sigs count="1">
         <cert index="4"  key="3082030d30820…37cf0aa3a31243230f4e48f"/>
      </sigs>
</package>
```

PackageManager `packages.xml` file


App
Sandbox
Payment APP
Applications
Contacts, Phone, Browser …
Application Framework (Java)
Package, Window, Location manager …
Android Native Libraries
(C/C++)
SSL, SQLite, libutils, libstagefright …
Android Runtime
Core Libraries (Java)
Dalvik VM
Linux Kernel
Device drivers, Power, Memory and Process Management
NFC Controller
Other Hardware

**Why does the security of Android application matter for PCI PTS?**

# Application Authentication

All Android apps must be digitally signed prior to installation

- The developer signs the corresponding *.apk file using his/her digital certificate (i.e., the corresponding private key)

Android uses the digital certificate as a means of

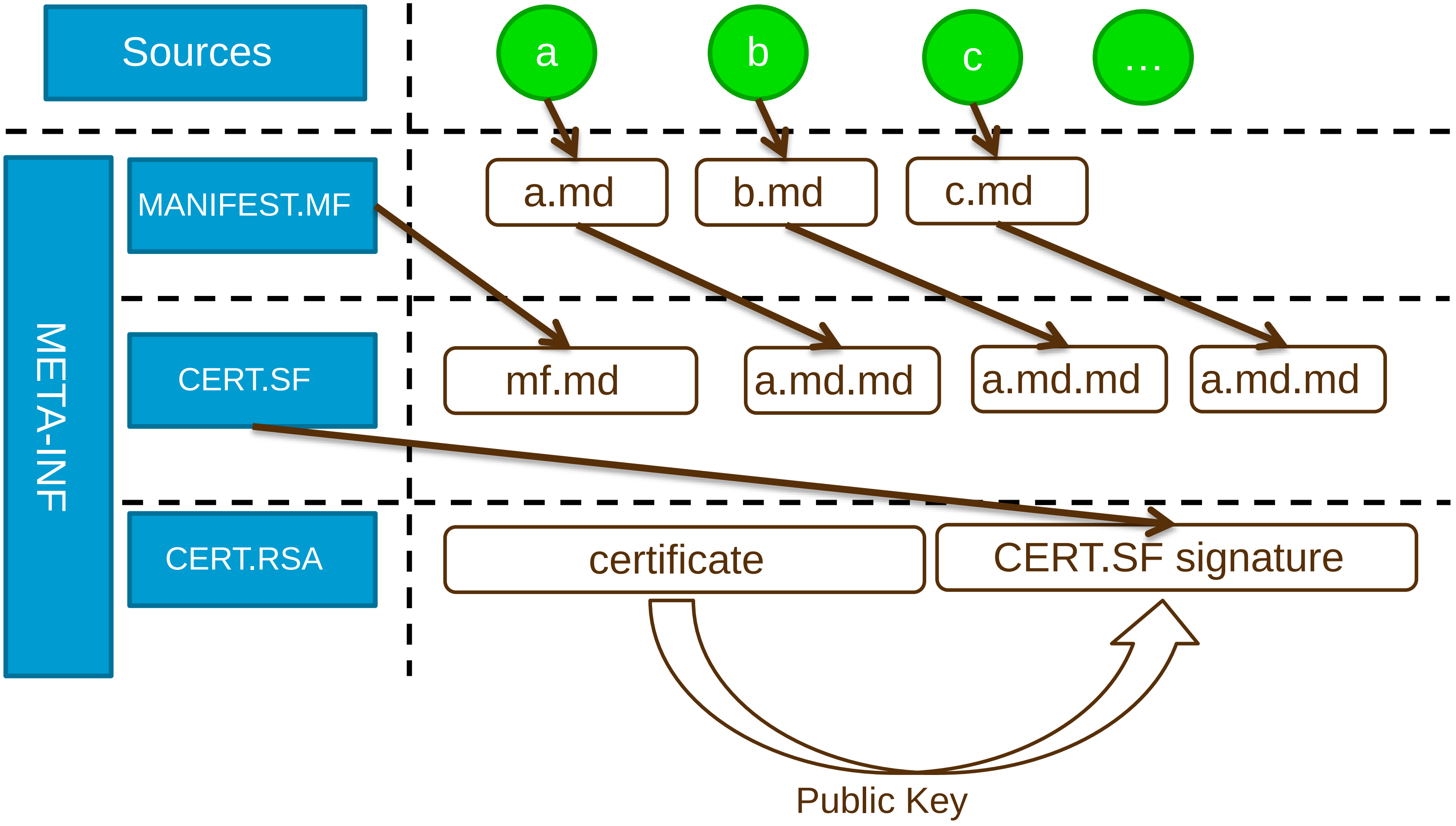
- Identifying the developer of an application
  - Used to ensure the authenticity of future application updates (same origin policy)

- Establishing trust relationships between applications
  - Applications signed with the same certificate can share, for example, the user ID (i.e., file system resources) and runtime process

# App authentication

- Each app (apk) is signed by developer and signature verified by install time.
- System apps are signed by set of *platform* keys
- No CAs (and no code-signing certificate from Google)
- Malicious app untraceable to author (however need to pay market fee to sell)
- Apps with the same UID must have been signed with the same key

# Application Signing



# APK Meta Information (META-INF/)

META-INF/ files hold necessary information (crypto hashes, certificate and signature) to verify the package integrity

- The signing process is based on cryptographic hash functions (e.g., SHA1) and public-key cryptography

```
more CERT.SF
SHA1-Digest-Manifest:
bFgRd0zf0ZHRZOr71smRiPIoo+I=

Name: res/drawable-xhdpi/ic_launcher.png
SHA1-Digest: iXOQFkCAFFovdXQunW5Lj2sge4k=
```

```
more MANIFEST.MF

Name: res/drawable-xhdpi/ic_launcher.png
SHA1-Digest: vWrq4ApK74D3ktrs7+elAA8A1a8=

Na
SH

…
Na
SH

Na
SH
```

```
keytool -printcert -file CERT.RSA

Owner: CN=Android Debug, O=Android, C=US
Issuer: CN=Android Debug, O=Android, C=US
Serial number: 33ac9dfa
Valid from: Thu Jan 09 16:41:38 CET 2014 until: Sat Jan 02 16:41:38 CET 2044
Certificate fingerprints:
         MD5:  A3:12:E2:09:D7:AF:88:AC:6F:0A:BF:C8:79:82:4A:86
         SHA1: 95:1C:B1:D0:4E:3D:57:FA:89:39:54:27:35:DC:25:53:8B:62:24:D0
         SHA256: A3:32:FB:6F:4D:37:5D:A3: ... :07:26:03:6C:EA:91:06:FC:9D
         Signature algorithm name: SHA256withRSA
         Version: 3
```

# Application Upgrade

- Applications can register for auto-updates
- Applications must have been signed with the same (private) key.
- No additional permissions should be added
- Install location is preserved

# Permissions

- **Can be**
  1) **Built-in**
  2) **App-defined**

- **Used to protect:**
  - **System sensitive API from app calls**
  - **Apps from each other (mutually distrusting)**

- **Managed by *package manager* *(/data/system/packages.xml)***
- **Permission level**

  ***Normal***
  ***Dangerous***
  ***Signature***
  ***SignatureOrSystem***

# Permissions - Android Manifest

Applications do not have default permissions
Each application can declare the requiRED permissions”.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <activity android:name="com.example.project.myActivity“
             android:permission=“com.myapp.mypermission”
             android:exported=“true”>
          <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
              </intent-filter>
                        . . .
        </activity>
        <activity> ….. </activity>
     <uses-permission android:name="android.permission.SEND_SMS" />
      <uses-permission android:name="android.permission.INTERNET" />  . . .
    </application>
</manifest>
```

# Permissions

- Android’s permission only restricts components from accessing resources
  - E.g. an application needs the READ_CONTACTS permission to read the user’s address book

- If a public (exported) component doesn’t explicitly declare any access permission, Android allow any application to access it.

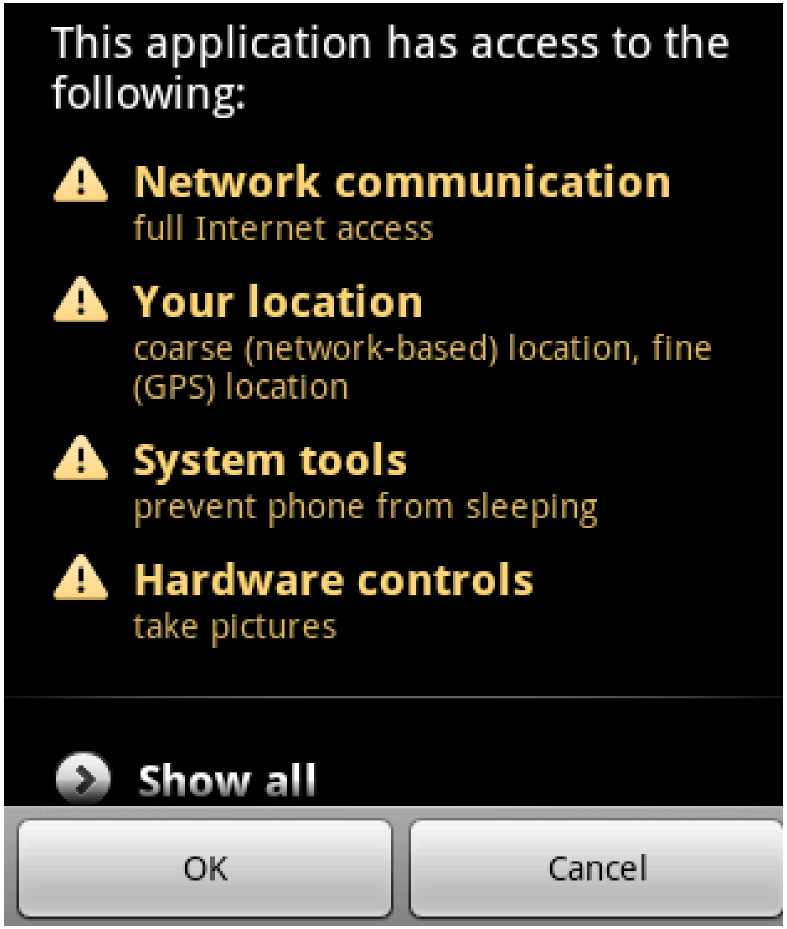
- Custom permissions are declared in the application manifest with the `<permission>` tag.
  - What does this mean to terminal vendors?

# Android Permission Model

- Permissions are the core concepts in the Android security to control the access from one application component to another.

- Before 6.0 All permissions are set at installation time and can’t change until the application is reinstalled.

- Starting from Android 6.0 (Marshmellow) application permissions can be individually toggled ON/OFF by the user.



This application has access to the following:

⚠ **Network communication**
full Internet access

⚠ **Your location**
coarse (network-based) location, fine (GPS) location

⚠ **System tools**
prevent phone from sleeping

⚠ **Hardware controls**
take pictures

# SELinux

- Used from Android 4.3 up
- Introduces MAC
- Default denial access control based on minimal privilege

- Modes → Permissive (denied permissions are logged but **not enforced**)
  → Enforcing (denied permissions are logged and enforced)
- A 4.3 Permissive mode ➔ A 4.4 Partially enforcing ➔ A 5.0 Full enforcing mode

- Defines Domains (Set of identically labeled processes) to be treated equally by security policy
- Have to create & maintain security policies

# Rooting
## (Entry points)

- **Bootloader ➔** if unlocked then flash arbitrary ROM into device. Locked bootloader: either commands are encrypted with SBK (Secure Boot Key) or image is signed

- **Recovery OS ➔** for reformatting of data partition, flash ROM. (update.zip in root of microSD card, signed by vendor). Root user can replace recovery with custom one.

- **ADB** (Android Debug Bridge) ➔ can run shell on device. Shell runs as root if *ro.secure==0* or as unprivileged user if *ro.secure==1*. Value of *ro.secure* comes from *default.prop* in root directory on boot-time which comes from a partition in internal storage. To write to partition one needs root access.

- *su/sudo* ➔ Needed to achieve privilege escalation in Linux (only *su* can run setuid(0)) to change process privilege. Stock OEM ROMs don’t come with *su/sudo*!

- **Conclusion: Rooting can be**
- **1) Easy -** Vendor leaves bootloader unlocked or *ro.secure==0 (ADB method)*
- ***2) Hard* -** Achieved by exploiting vulnerabilities in “root” level running processes (kernel).

# Questions?