Android's security architecture

Nikolay Elenkov

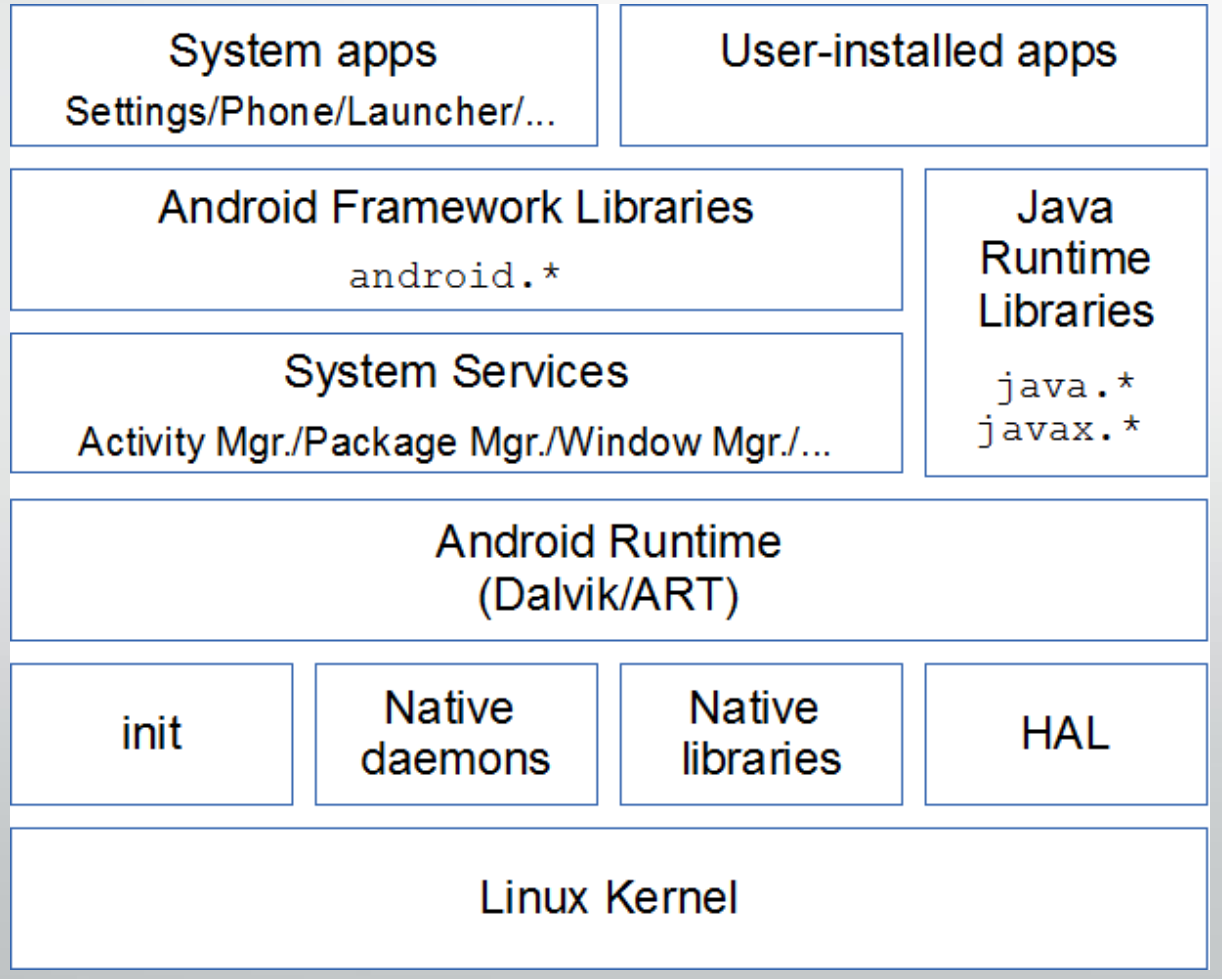
Android Security Symposium, Sep 2015

Vienna

Agenda

- Android's architecture and security model
- Package management
- Permissions
- SELinux
- User management
- Cryptography, PKI, and credential storage
- Enterprise security and Android for Work
- Device security and verified boot
- NFC and secure elements

Android's architecture



Android + TEE

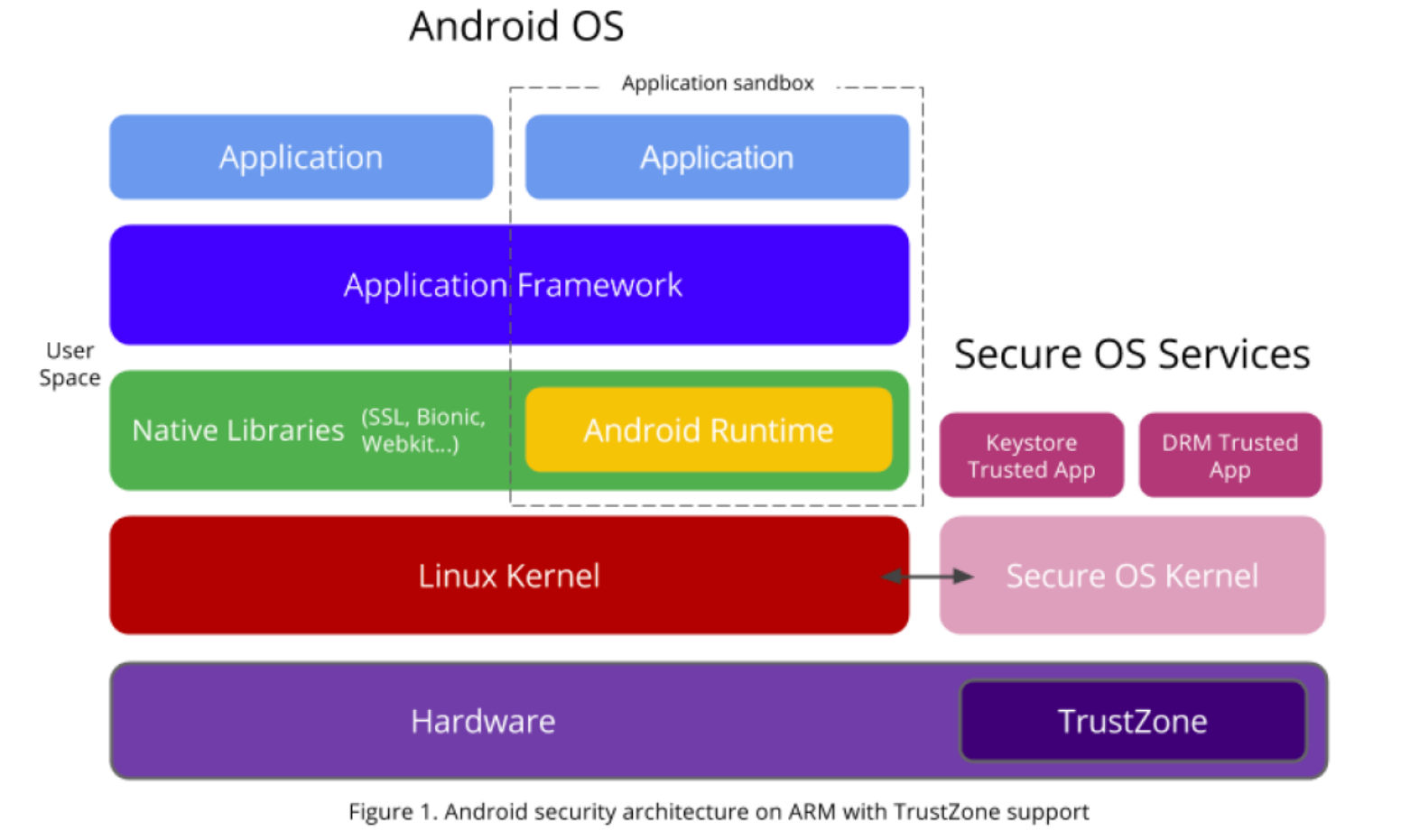


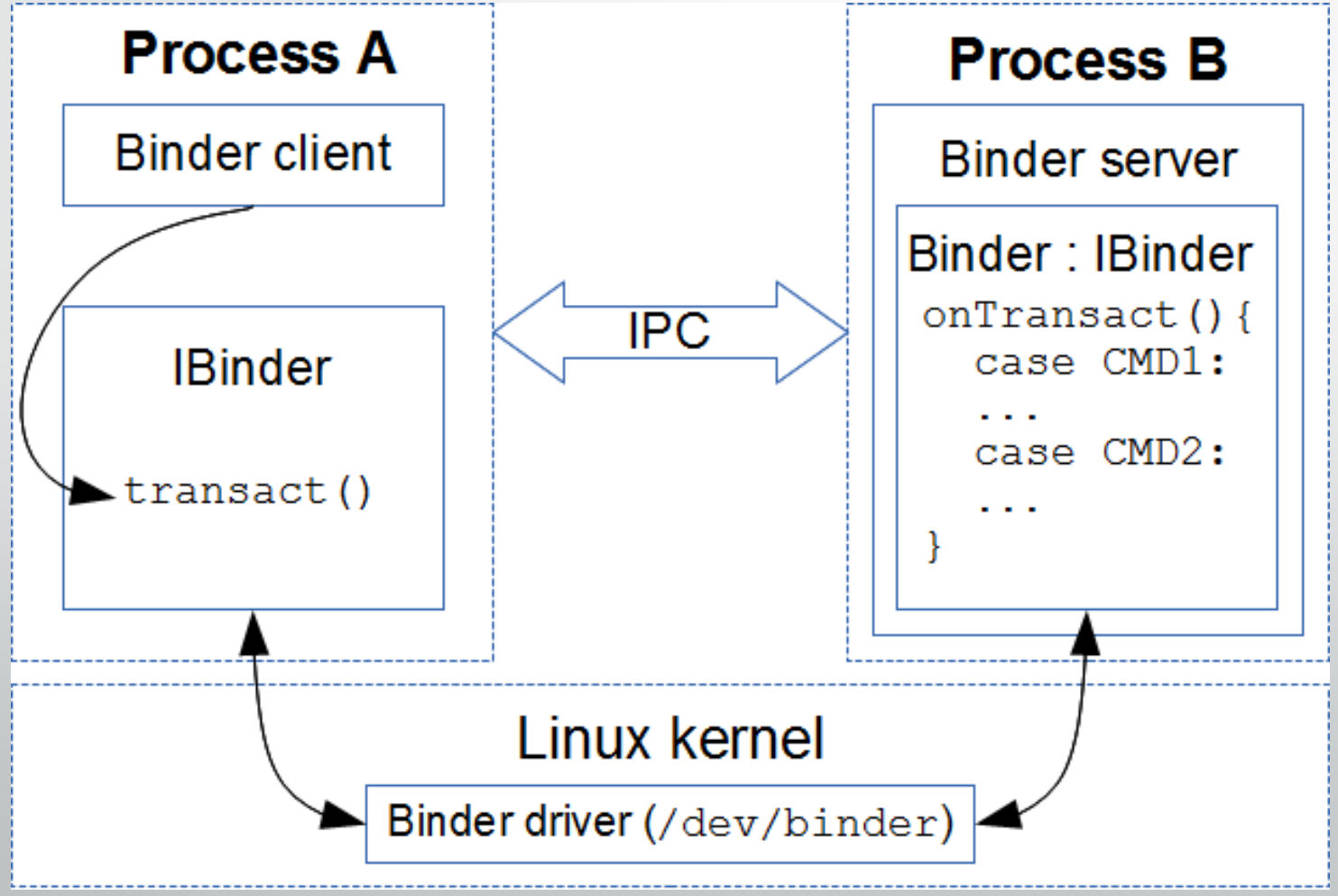
Figure 1. Android security architecture on ARM with TrustZone support

From "Android for Work Security white paper"
https://static.googleusercontent.com/media/www.google.co.jp/en/US/work/android/files/android-for-work-security-white-paper.pdf

Security model

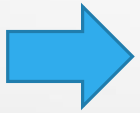
- Kernel-based application sandbox
  - DAC (UID, GID-based access control) and MAC (SELinux type enforcement)
  - Dedicated, per-application UIDs
- Secure IPC (local sockets, Binder, intents)
- System services running with reduced privileges
- Code signing
  - Application packages (APKs)
  - OS update packages (OTA packages)
- Android permissions
  - System and custom (application defined)
  - Required to access:
    - System resources/hardware
    - Personal information (contacts, email address, location, etc.)
    - Exported application components

Binder IPC


Process A
Binder client
IBinder
transact()
IPC
Process B
Binder server
Binder : IBinder
onTransact(){
case CMD1:
...
case CMD2:
...
}
Linux kernel
Binder driver (/dev/binder)

Package installation

```
com.example.app.apk/
    |-- AndroidManifest.xml
    |-- classes.dex
    |-- resources.arsc
    |-- lib/
    |   |
    |   `-- armeabi-v7a/
    |       `-- libapp.so
    |-- META-INF/
    |   |-- CERT.RSA
    |   |-- CERT.SF
    |   `-- MANIFEST.MF
    `-- res/
        |-- drawable/
        |-- layout/
        `-- xml/
```



- Code and resources (common)

```
/data/app/com.example.app/
    |-- lib/arm/libapp.so
    |-- oat/arm/base.odex
    `-- base.apk
```

- Data (per user)

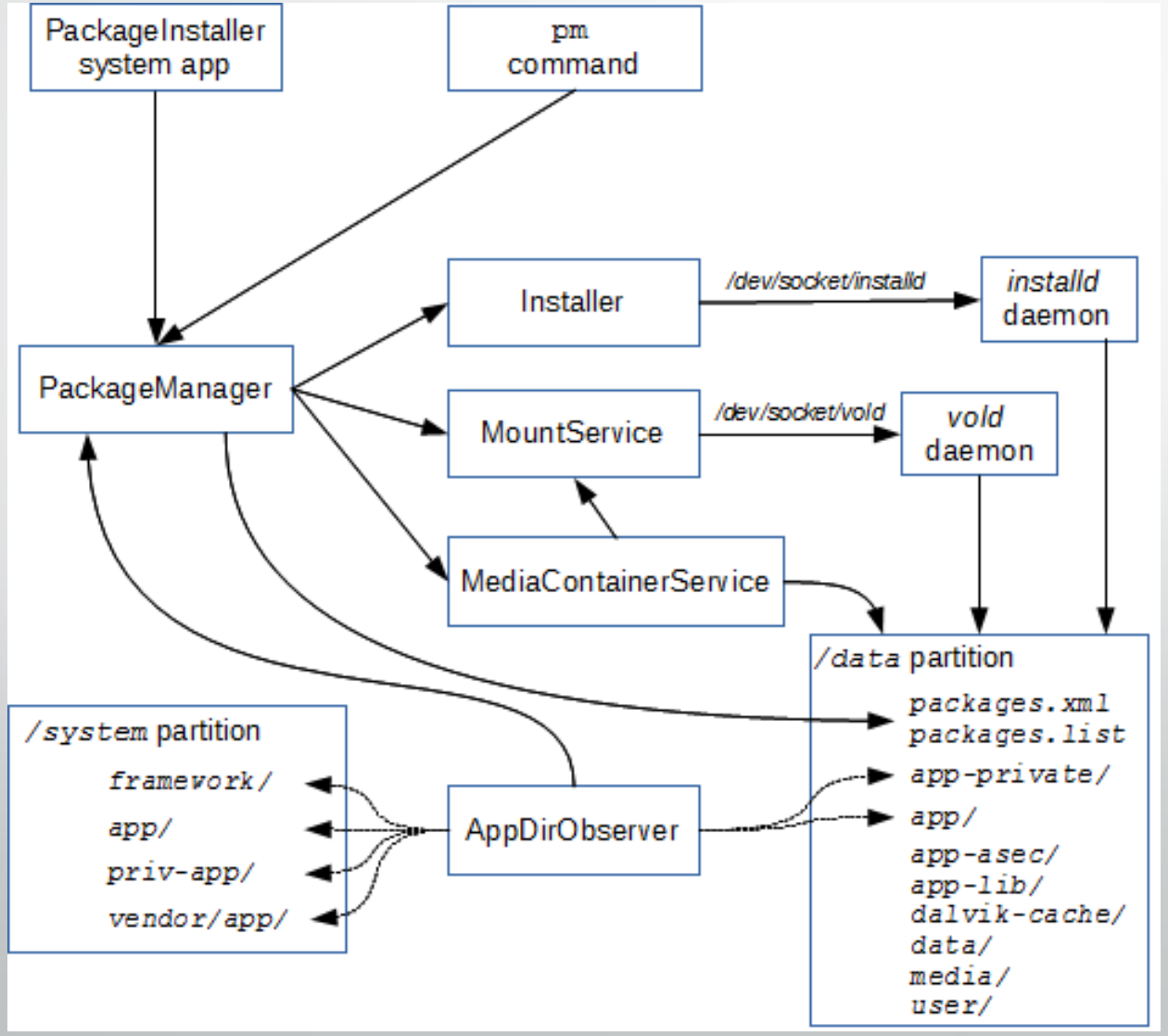
```
/data/user/0/com.example.app/
    |-- files/
    |-- databases/
    |-- shared_prefs/

/data/user/1/com.example.app/
…
```

Package data directories

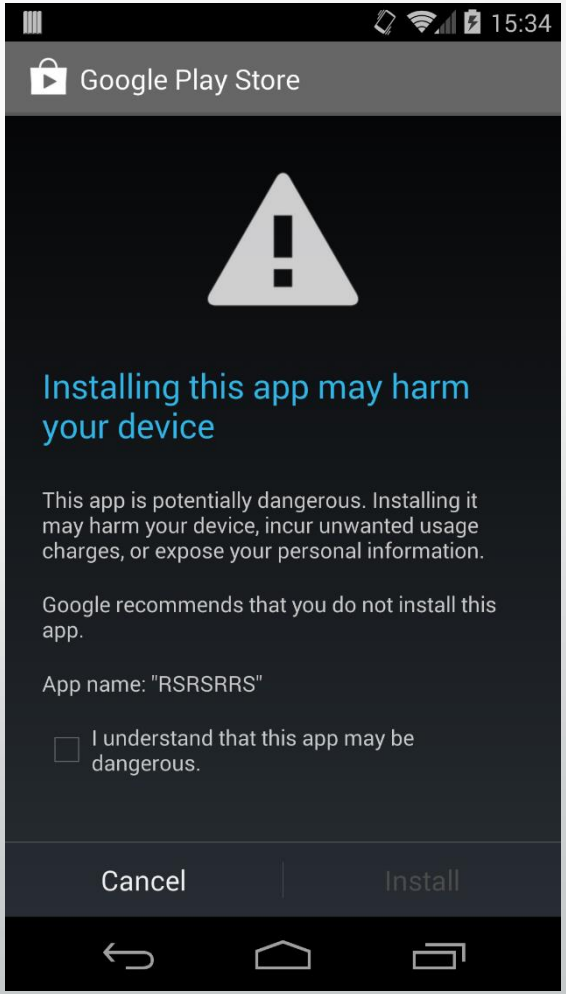
```
# ls -lZ /data/user/0/
drwxr-x--x u0_a185  u0_a185    u:object_r:app_data_file:s0:c512,c768 at.mroland.android.apps.nfctaginfo
drwxr-x--x bluetooth bluetooth u:object_r:bluetooth_data_file:s0 com.android.bluetooth
drwxr-x--x system    system    u:object_r:system_app_data_file:s0 com.android.keychain
drwxr-x--x u0_a4    u0_a4      u:object_r:app_data_file:s0:c512,c768 com.android.providers.calendar
drwxr-x--x system    system    u:object_r:system_app_data_file:s0 com.android.providers.settings
drwxr-x--x radio     radio     u:object_r:radio_data_file:s0 com.android.providers.telephony
drwxr-x--x u0_a5    u0_a5      u:object_r:app_data_file:s0:c512,c768 com.android.providers.userdictionary
drwxr-x--x u0_a27   u0_a27     u:object_r:app_data_file:s0:c512,c768 com.android.proxyhandler
drwxr-x--x u0_a115  u0_a115    u:object_r:app_data_file:s0:c512,c768 com.bria.voip
drwxr-x--x u0_a190  u0_a190    u:object_r:app_data_file:s0:c512,c768 com.codebutler.farebot
drwxr-x--x u0_a142  u0_a142    u:object_r:app_data_file:s0:c512,c768 com.csipsimple
```

Package management components



Advanced package management

- Updating system apps
  - /system/app/ ➔ /data/app/
- Encrypted packages
- Forward locking
  - Installing in encrypted container
  - Mainly for paid apps (DRM)
- Package verification
  - Verification agents
  - Default agent in Google Play
  - Sends APK details to Google



Code signing

- For application packages (APKs)
  - Self-signed X.509 certificates, treated as binary blobs
  - Not using PKI (no certificate chain building)
  - Individual signature for each file included in APK
  - Signing certificate == package identity
  - Package updates require same certificate
  - Certificate required to grant signature permissions or shared user ID
- For update packages (OTAs)
  - Modified ZIP format
  - Signature in ZIP comment, over whole file (excluding comment)
  - Verified by OS and recovery
- System images may also be signed (required as of 6.0)

APK code signing example

- APK signature file (META-INF/CERT.SF)

```
Signature-Version: 1.0

Created-By: 1.0 (Android SignApk)

SHA1-Digest-Manifest:
Hh+AqEL1RMpxY+SpzJRpv4pcyG4=


Name: classes.dex

SHA1-Digest-Manifest:
ikCuogTuKUl4NoGNlTW9QOmxeEk=


Name: res/anim/slide_left_in.xml

SHA1-Digest-Manifest:
VBc3lMcURseVYOwtwkARy4u5n9I=
```

- APK signature block (META-INF/CERT.RSA)

```
$ jarsigner -keystore platform.keystore -
verify -verbose -certs Calendar.apk

smk   1168568 classes.dex

  X.509, EMAILADDRESS=android@android.com,
(testkey)

  [certificate is valid from 2/29/08 to
7/17/35]

smk      428 res/anim/slide_left_in.xml

  X.509, EMAILADDRESS=android@android.com,
(testkey)

  [certificate is valid from 2/29/08 to
7/17/35]


  s = signature was verified

  m = entry is listed in manifest

  k = at least one certificate was found in
keystore
```

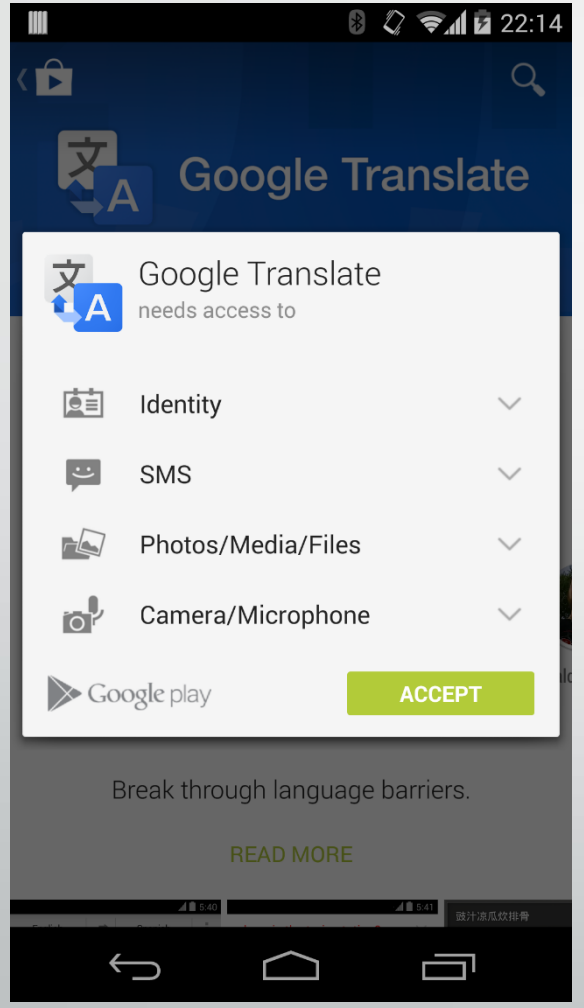
Permissions (1)

- Permission: ability to perform particular operation
  - Could be regarded as a form of MAC
- Enforced at different levels
  - Kernel (e.g., `INTERNET` permission)
  - Native service level
    - Usually mapped to groups (`READ_EXTERNAL_STORAGE` ➔ `sdcard_r`)
  - Framework level (`PackageManager` and `ActivityManager`)
    - Dynamic: `checkUidPermissions()`, mainly services
    - Static: intents, content providers
- Assignment
  - Traditionally at install time
  - Also at runtime since Android 6.0

Permissions (2)

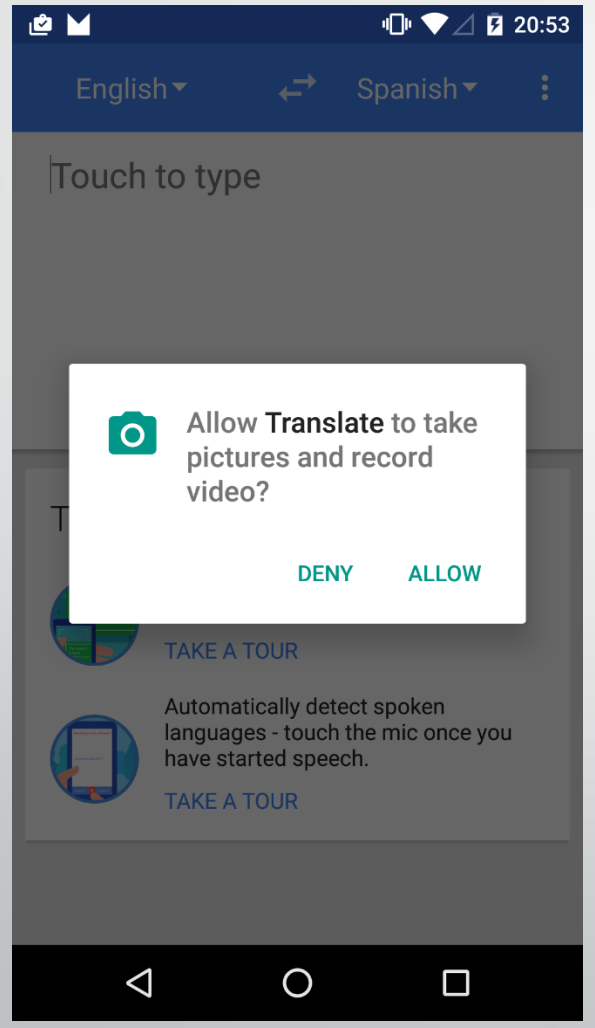
- Protection levels
  - `normal`
  - `dangerous`
  - `signature`
  - `signatureOrSystem (signature|privileged)`
- System permissions
  - `android` package, defined in `framework-res.apk`
- Custom permissions
  - Defined by applications
- Shared user ID
  - Apps with same signature can run as same UID
  - Each app receives union of permissions declared by shared user ID
- Permission groups: related permissions
  - `CONTACTS`, `STORAGE`, `LOCATION`

Install-time permissions



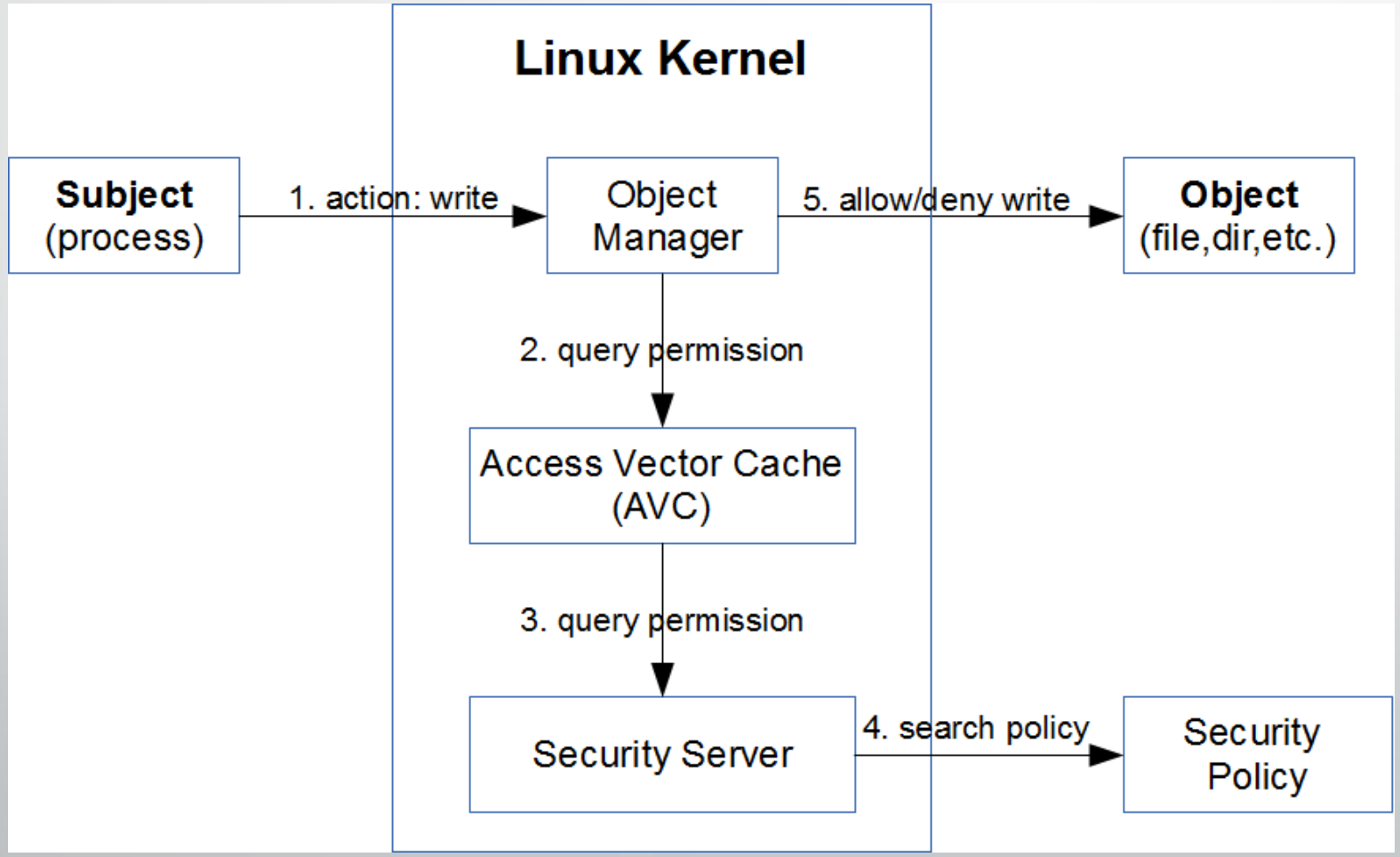
- All permissions granted at install time
- `dangerous` permissions require user confirmation
- No runtime checks required
- Once granted, permissions cannot be revoked
  - Except for developer permissions
- Fine grained
- Granted for all users on device
- Stored in `/data/system/packages.xml`

Runtime permissions



- Need to prompt for `dangerous` permissions at runtime
- Can be revoked at any time
- Granted/revoked by permission group
  - No prompt for other permission from same group
  - Coarse grained
- Managed per app, per user
  - `/data/system/users/0/runtime-permissions.xml`
- Some permissions cannot be revoked
  - `FLAG_PERMISSION_POLICY_FIXED`
  - `FLAG_PERMISSION_SYSTEM_FIXED`
- Managed by device owner (via `DevicePolicyManager`)
  - `setPermissionGrantPolicy()`
  - `setPermissionGrantState()`

SELinux


Linux Kernel
Subject
(process)
1. action: write
Object
Manager
5. allow/deny write
Object
(file,dir,etc.)
2. query permission
Access Vector Cache
(AVC)
3. query permission
Security Server
4. search policy
Security
Policy

SELinux policy example

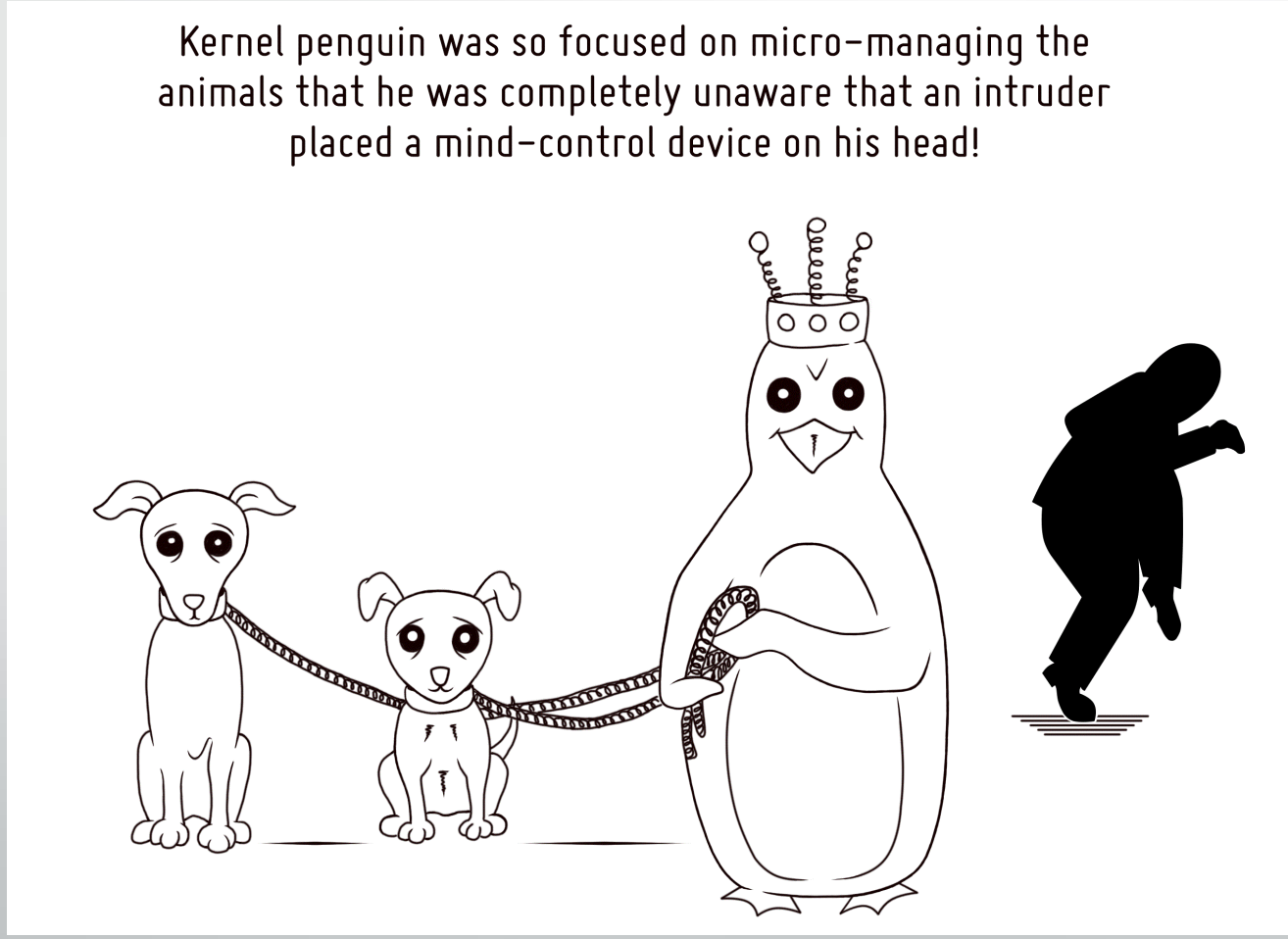
```
type keystore, domain;
type keystore_exec, exec_type, file_type;

# keystore daemon
init_daemon_domain(keystore)
typeattribute keystore mlstrustedsubject;
binder_use(keystore)
binder_service(keystore)
allow keystore keystore_data_file:dir create_dir_perms;
allow keystore keystore_data_file:notdevfile_class_set create_file_perms;
allow keystore keystore_exec:file { getattr };
allow keystore tee_device:chr_file rw_file_perms;
allow keystore tee:unix_stream_socket connectto;
```

SELinux in Android (SEAndroid)

- Binder support (LSM hooks in kernel added)
- New `init` commands (`seclabel`, `restorecon`, …)
- Labelling for system properties
  - Based on rules in `property_contexts`
- Labelling application processes
  - All forked from `zygote`, cannot use domain transition
  - Security context derived based on rules in `seapp_contexts` file
- Middleware MAC (MMAC)
  - `seinfo` label set based on signing certificate
  - Rules defined in `mac_permissions.xml`

An alternative view…

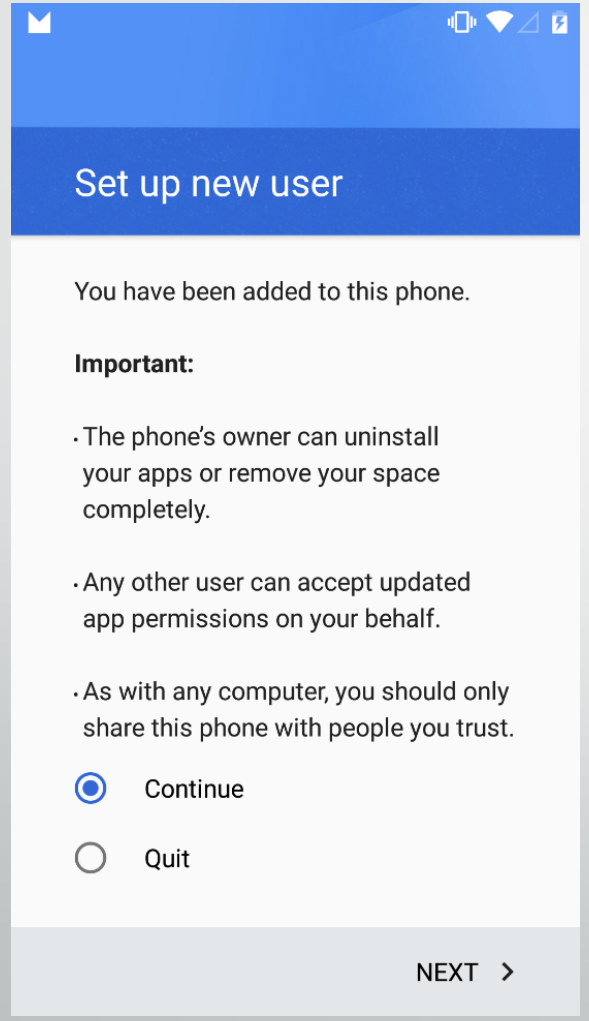


From https://grsecurity.net/~spender/pics/lost_selinux_coloring_book_page1.gif

Multi-user support

- Originally for tablets only, now for phones also (as of 5.0)
- Users are isolated by UID/GID
- Separate settings and app data directories
  - system directory: `/data/system/users/<user ID>/`
  - app data directory: `/data/user/<user ID>/<pkg name>/`
- Apps have different UID and install state for each user
  - app UID: `uid = userId * 100000 + (appId % 100000)`
  - shared applications: install state in per-user `package-restrictions.xml`
- External storage isolation

Types of users

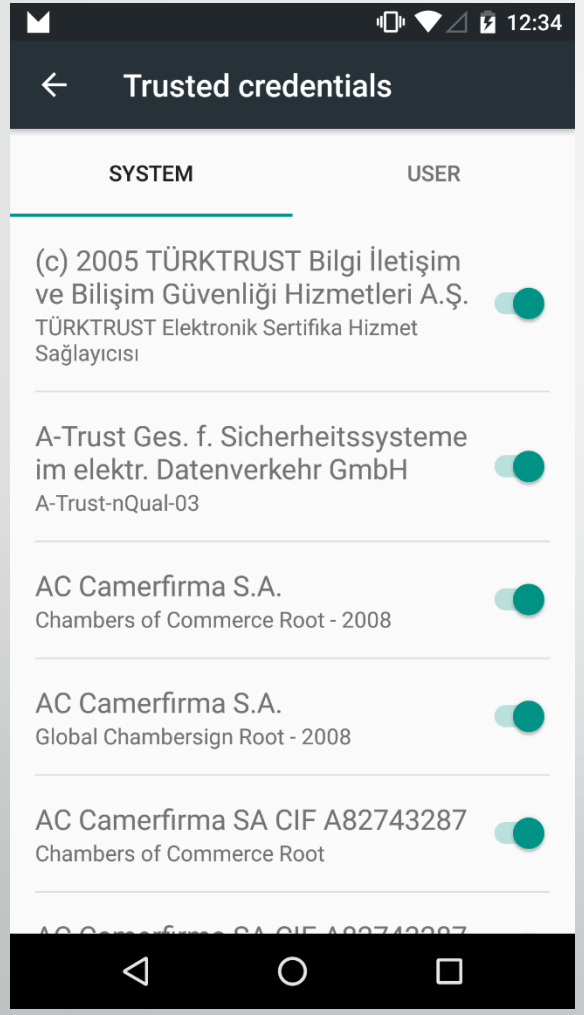


- Primary user (owner)
  - Full control over device
- Secondary users
- Restricted profile
  - Shares apps with primary user
  - Only on tablets
- Managed profile
  - Separate apps and data, but shares UI with primary user
  - Managed by Device Policy Client (DPC)
- Guest user
  - Temporary, restricted access to device
  - Data (session) can be deleted

Cryptography and SSL

- JCA provider architecture, multiple providers:
  - `Crypto`: From Apache Harmony
    - *SHA1PRNG* only, for backwards compatibility
  - `BC`: (modified) Bouncy Castle
  - `AndroidOpenSSL`: Open/BoringSSL based. Project name: *conscrypt*
    - Main provider
    - native code+JNI wrappers
  - `GmsCore_OpenSSL`: in Play Services, automatically updated
  - `AndroidKeyStore`: Generates unextractable keys managed by system keystore
    - RSA, EC, HMAC and AES (as of 6.0)
- SSLv3, TLS v1.0-v1.2 support: JSSE API, providers:
  - `HarmonyJSSE` (deprecated)
  - `AndroidOpenSSL`

Certificates and PKI



- Android-specific trust store
- Pre-installed trust anchors (‘trusted credentials’)
- User-installed trust anchors
  - Per user/profile
- Modified certificate chain building
  - Based on Bouncy Castle code
  - Dynamically updated certificate blacklists
  - Dynamically updated certificate pinning for Google sites

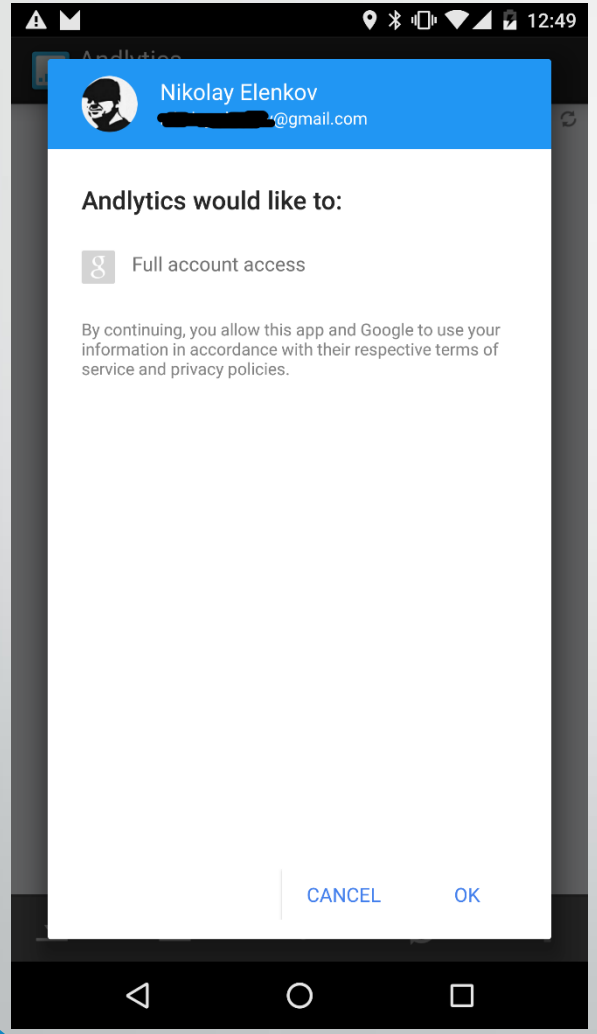
Network security

- WPA EAP2 Enterprise (802.11i)
  - EAP: EAP-TLS, EAP-TTLS, PEAP, EAP-SIM, EAP-AKA since Android 5.0
  - Integrates with system keystore
  - Integrates with Android for Work (device administration APIs)
- VPN
  - Legacy VPN: PTPP and IPSec
  - Always-on VPN: no network access until VPN is up
  - Per-user/profile VPNs
    - Dynamic routing/firewall rules
  - Per-application VPN since Android 5.0

Credential storage

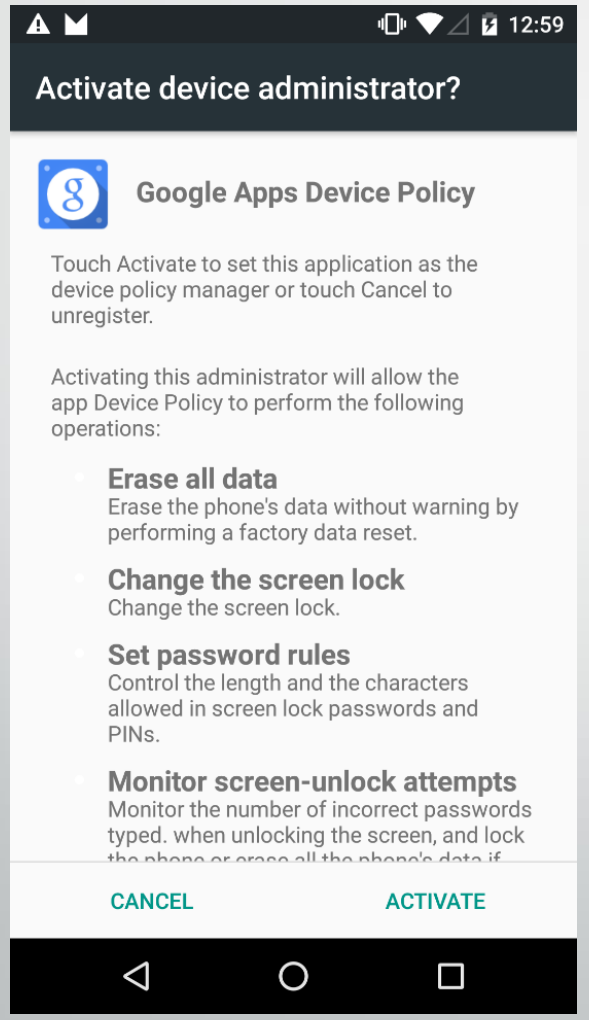
- System-managed, secure cryptographic key store
  - Unexportable keys
  - Remains secure even if main OS is compromised (if HW-backed)
- Implemented in the `keystore` system service
  - HAL interface (*keymaster*), hardware-backed implementations possible
  - Typically uses TEE (implemented using TrustZone) on ARM devices
- Provides framework APIs
  - `KeyChain` API
  - `KeyStore`
  - `KeyPairGenerator`, `KeyGenerator`

Online account management



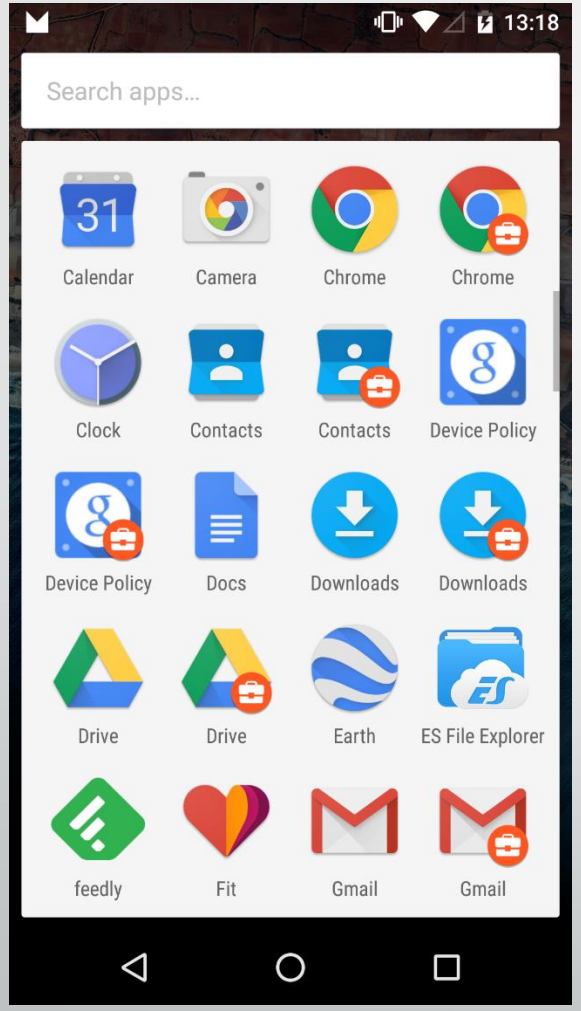
- System store for accounts, passwords, and authentication tokens
- `AccountManager` API
  - Pluggable architecture
  - Designed for passwords, not very flexible
- Token requests confirmed by user
  - One of the first runtime permissions
- Google accounts are special
  - Master token saved on first authentication
  - User can control access in their account page on Web
  - Supports 2FA (OTP only for now)

Device administration



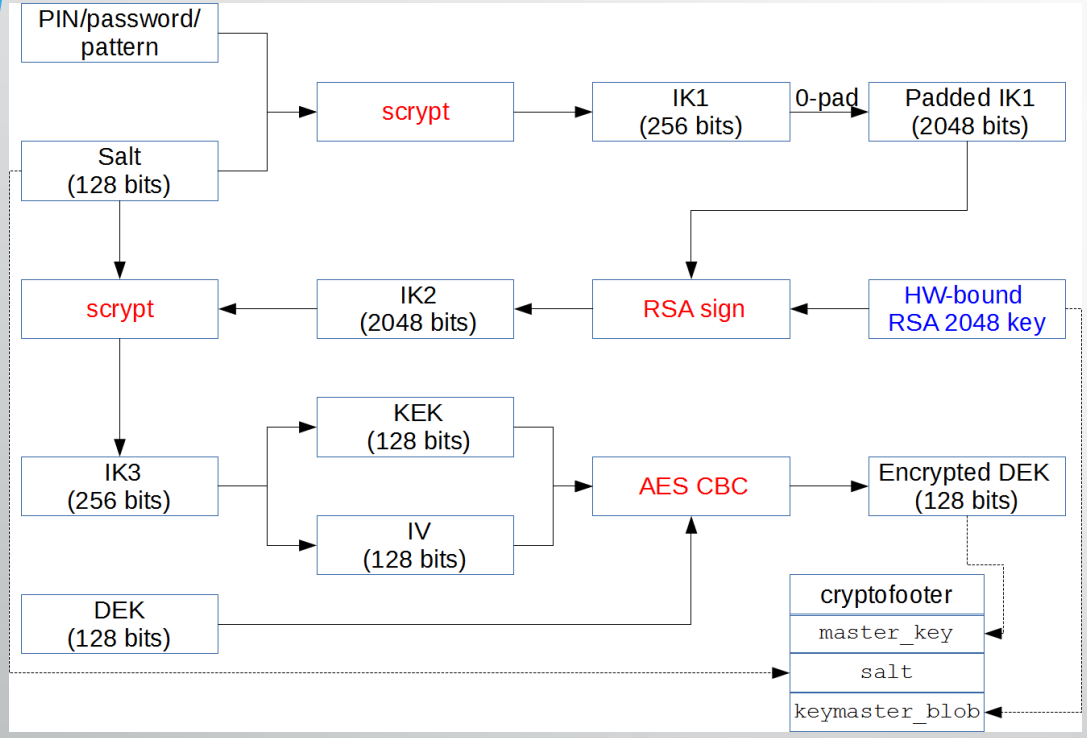
- Device security policy can be set by 'device administrator'
  - Password/PIN policy
  - Device lock/unlock
  - Storage encryption
  - Camera access
  - Much more control if version > 5.0
- Needs to be activated by user
- Cannot be directly uninstalled
  - Needs to be disabled first
- May be required to sync account data
  - MS Exchange (EAS)
  - Google Apps

Android for Work



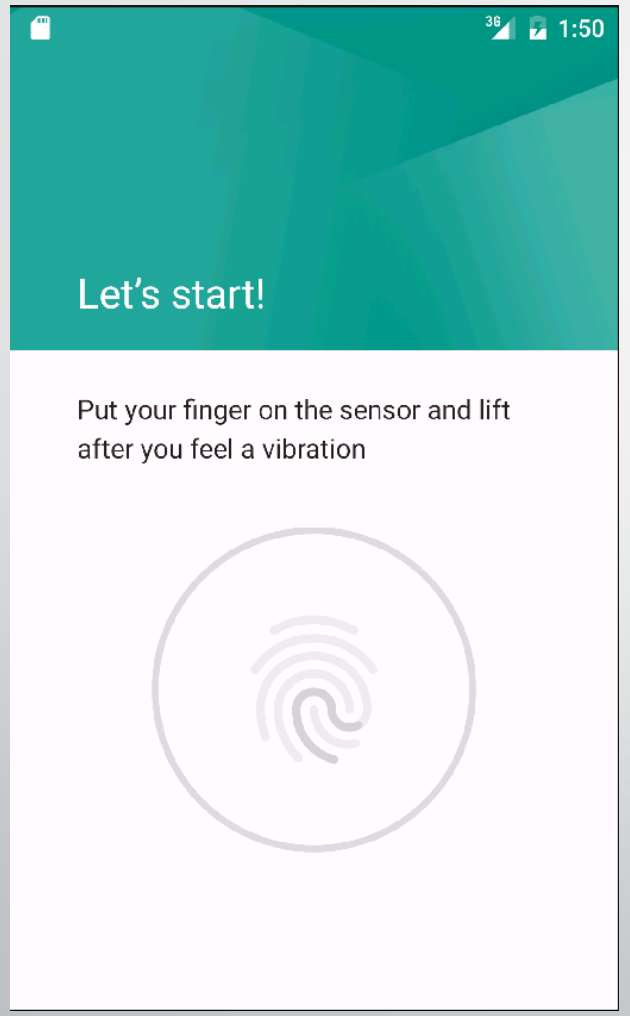
- Android > 5.0 supports a 'Work Profile'
  - Follows pre-defined managed provisioning flow
  - Managed by 'Profile Owner' device admin
  - Requires device encryption
- Separate apps and data
  - Can only install pre-approved apps
- But shares UI with primary user
  - Launcher/Notifications/Settings
- 'Device Owner' is a super-device admin
  - Installed when device is first initialized
  - Cannot be installed
  - Extra privileges
  - Scoped to device

Disk encryption



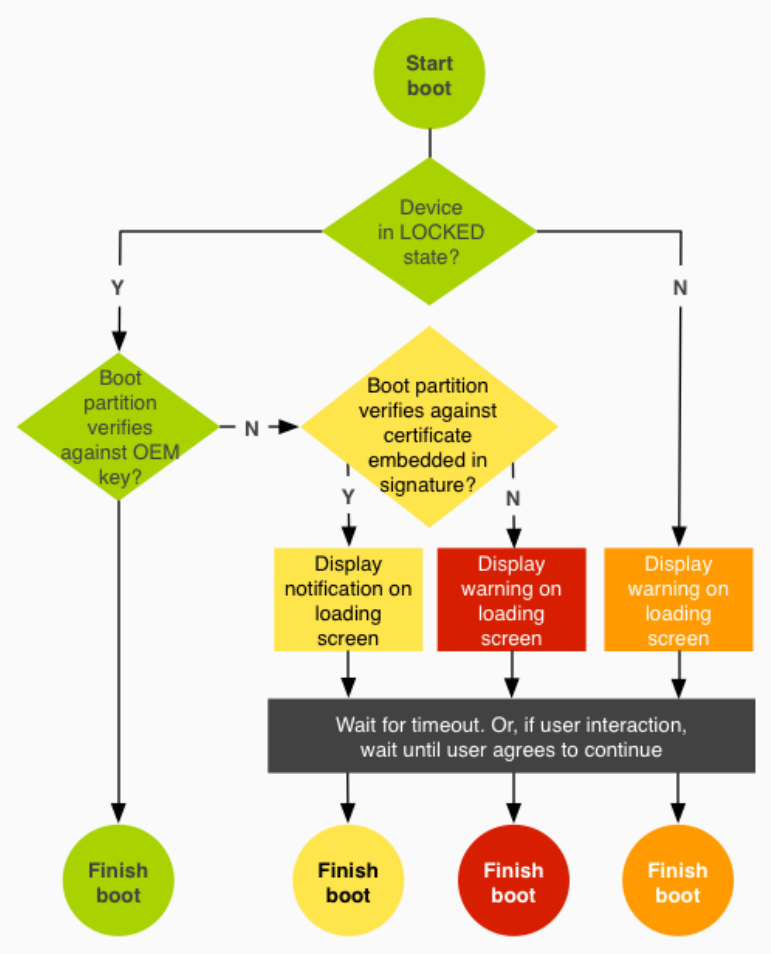
- Block device encryption, based on *dm-crypt*
  - *userdata* partition only
- AES 128 CBC and ESSIV:SHA256
- HW-accelerated encryption also supported
  - *dm-req-crypt*, AES XTS
- Master key (DEK) encrypted with AES 128
- KEK derived from PIN/password
  - *scrypt* algorithm
  - Protected by TEE key in Android > 5.0
- Optionally encrypt on first boot
  - `forceencrypt` flag, Android > 5.0
- File-based encryption (EXT4) coming soon?

Device security



- Lockscreen (keyguard)
  - Pattern (least secure)
  - PIN/Password
  - Stores hashes, uses Gatekeeper HAL since 6.0
- Smart Lock since 5.0
  - Trust agents
  - Extensible
  - Bluetooth, NFC, Location, Face (Google proprietary)
- Factory reset protection since 5.1
  - Google account info saved on `frp` partition
- Fingerprint since 6.0
  - Fingerprint HAL
  - Can be used for payment authorization, etc.

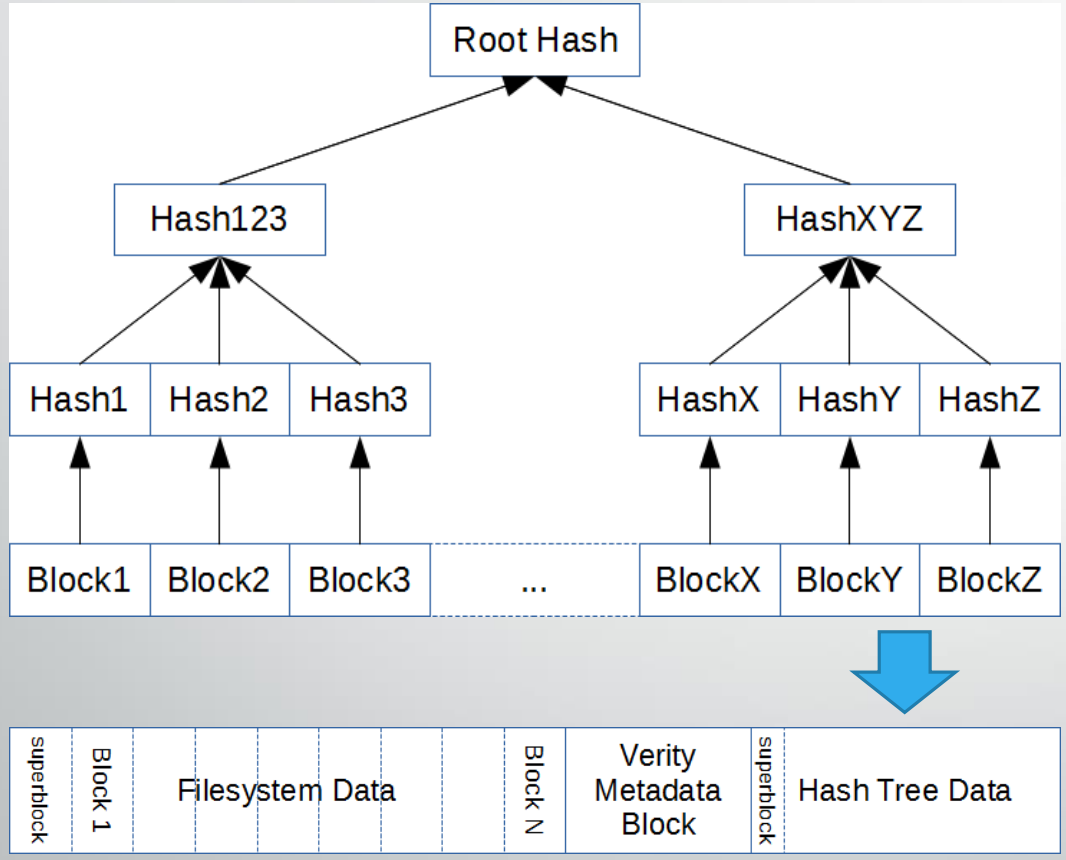
Verified boot



- Device software integrity based on HW root of trust
- Boot chain (simplified)
  - Verify bootloader using HW root of trust
  - Bootloader verifies *boot*/*recovery* partition
  - Kernel verifies *system* partition
- Device (bootloader) state
  - `LOCKED/UNLOCKED`
  - Allows custom (non-OEM) keys
- Boot state
  - `GREEN/YELLOW/ORANGE/RED`
  - Doesn't stop boot, only shows warning

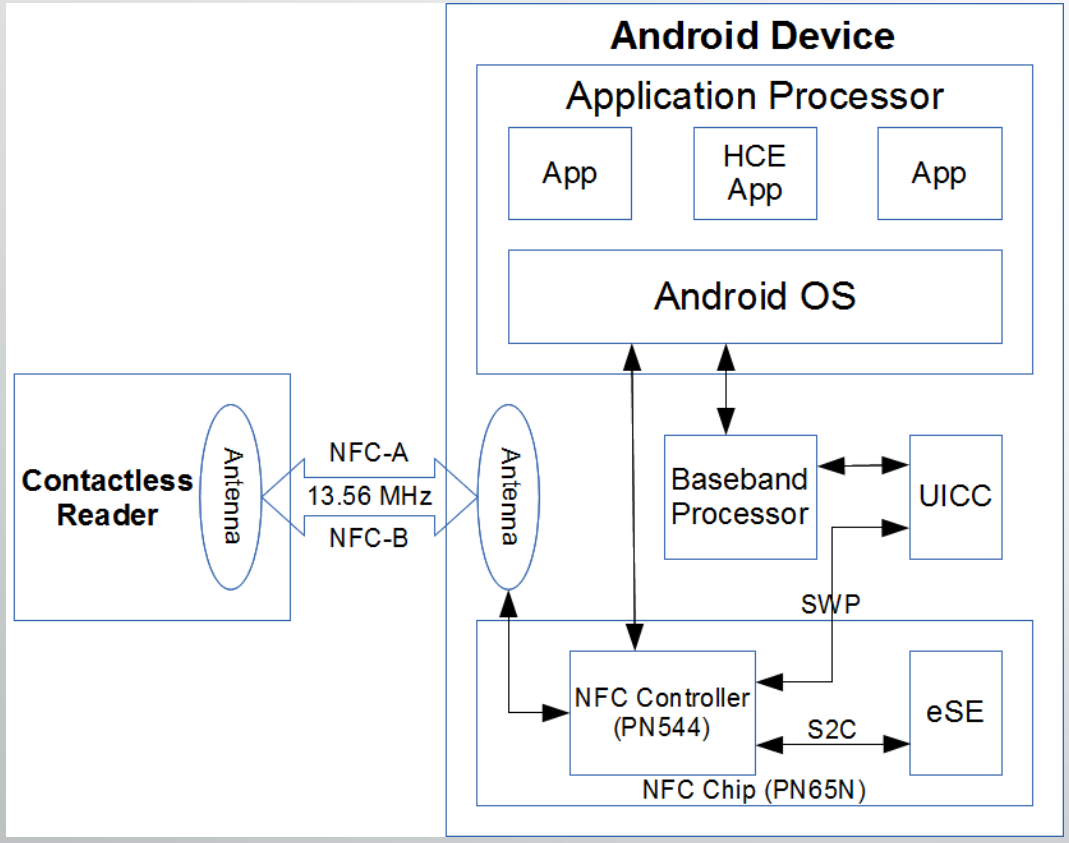
From "Verifying Boot",
https://source.android.com/devices/tech/security/verifiedboot/verified-boot.html

dm-verity



- *dm-verity*: transparent integrity checking for block devices
- Read error if block integrity check fails
- Useful for read-only partitions like *system*
- Requires block-based OTA updates
- Kernel needs to be trusted (verified boot)
- Stateful in Android 6.0
  - Default is *enforcing* mode
  - Falls back to *logging* mode if metadata cannot be verified
  - State saved in dedicated metadata partition
  - Does not stop boot, only shows warning

NFC and secure elements



- Near Field Communication (NFC)
  - Reader/write mode (RW)
  - Peer-to-peer mode (P2P)
  - Card emulation mode (CE)
    - Secure Element (SE), since 2.3
    - Host-based CE (HCE), since 4.4
- Secure Elements
  - UICC (SIM)
  - ASSD (microSD)
  - Embedded SE (eSE)
- APIs
  - Telephony APIs (restricted)
  - OpenMobile API (SEEK)
  - Android HCE (`HostApduService`)

References (Web)

- Official (Android documentation)
  - https://source.android.com/devices/tech/security/enhancements/
  - https://developer.android.com/preview/api-overview.html#afw
  - https://developer.android.com/about/versions/android-5.0.html#Enterprise
  - https://source.android.com/devices/tech/security/index.html
- Community
  - http://www.droidsec.org/wiki/ (Droidsec Wiki)
  - https://plus.google.com/communities/118124907618051049043 (Android Security G+ Community)
  - https://forum.xda-developers.com/general/security (XDA Security Forum)
- Mobile security companies
  - https://www.nowsecure.com/blog/ (NowSecure, formerly viaForensics)
  - https://bluebox.com/blog/business/ (Bluebox)
  - https://labs.mwrinfosecurity.com/publications/ (MWR InfoSecurity)

References (Books)

- The Mobile Application Hacker's Handbook, Wiley, 2015
- Android Internals, Jonathan Levin, 2015
- Android Hacker's Handbook, Wiley, 2014
- Android Malware and Analysis,Auerbach,2014
- Android Security Internals, No Starch, 2014
- Embedded Android, O'Reilly, 2013