

# Android Platform Security

*Mobile Security 2021*

Johannes Feichtner  
[johannes.feichtner@iaik.tugraz.at](mailto:johannes.feichtner@iaik.tugraz.at)

# Outline

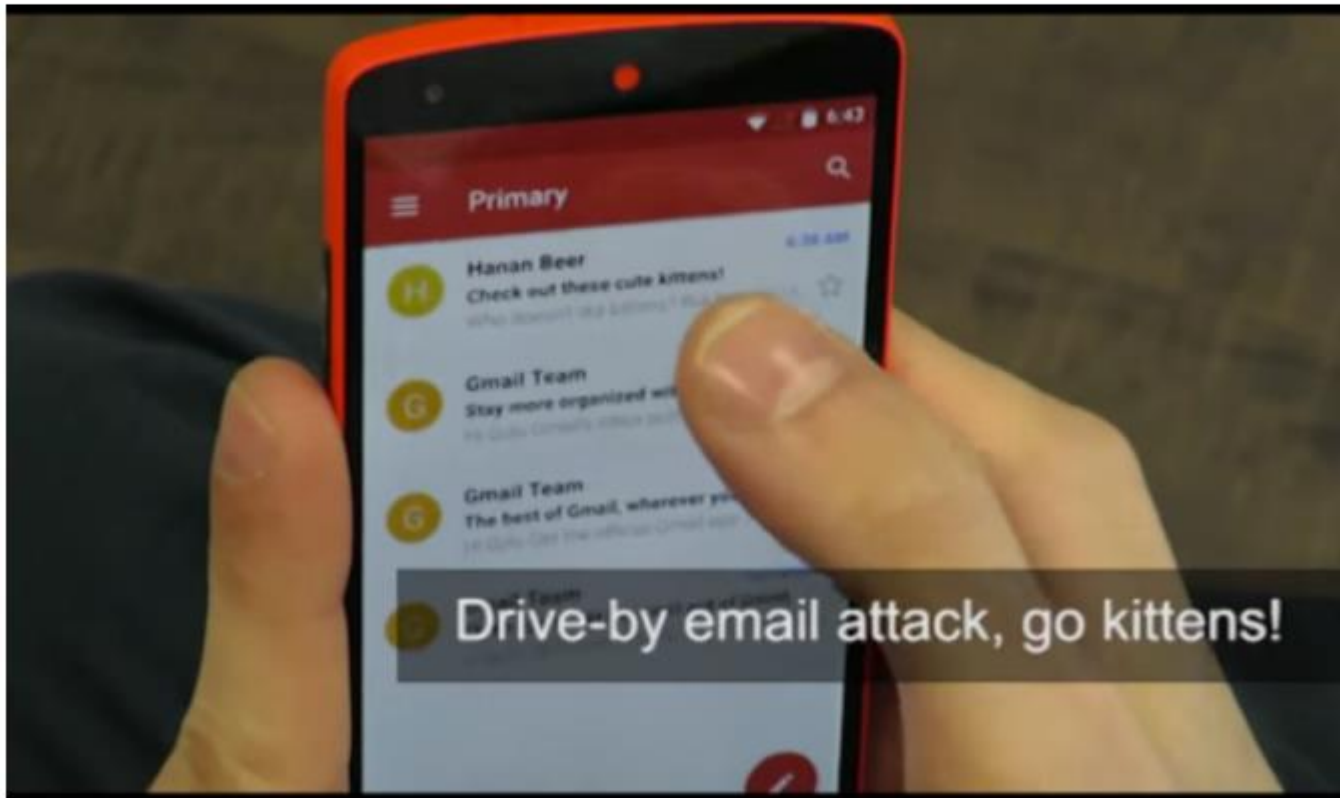
- Low-level System Security
  - Verified Boot & dm-verity
- Encryption System
  - Full Disk Encryption
  - File-based Encryption
- Android OS Security
  - Architecture & Sandbox
  - SELinux



# 275 million Android phones imperiled by new code-execution exploit

Unpatched "Stagefright" vulnerability gives attackers a road map to hijack phones.

DAN GOODIN - 3/18/2016, 9:26 PM



Source: <https://goo.gl/9fgYSc>



## What?

Bugs in Android's libstagefright and libutils

## How?

- Attacker embeds shellcode in harmless multimedia file
- Message is downloaded (e.g. via MMS)
- Exploit is executed

## Result

- Attacker can execute any code on remote device



# Serious flaw in WPA2 protocol lets attackers intercept passwords and much more

KRACK attack is especially bad news for Android and Linux users.

DAN GOODIN - 10/16/2017, 6:37 AM



## What?

Android can be tricked into using an all-zero encryption key for WPA/WPA2 WiFi communication

## How?

- Attacker resends message of 4-way handshake to device
- Real encryption key is replaced with zero key

## Result

- Attacker can intercept and manipulate traffic from device

# Billions of devices imperiled by new clickless Bluetooth attack

BlueBorne exploit works against unpatched devices running Android, Linux, or Windows.

DAN GOODIN - 9/12/2017, 3:00 PM



## What?

Implementation flaws in common Bluetooth stacks enable remote code execution

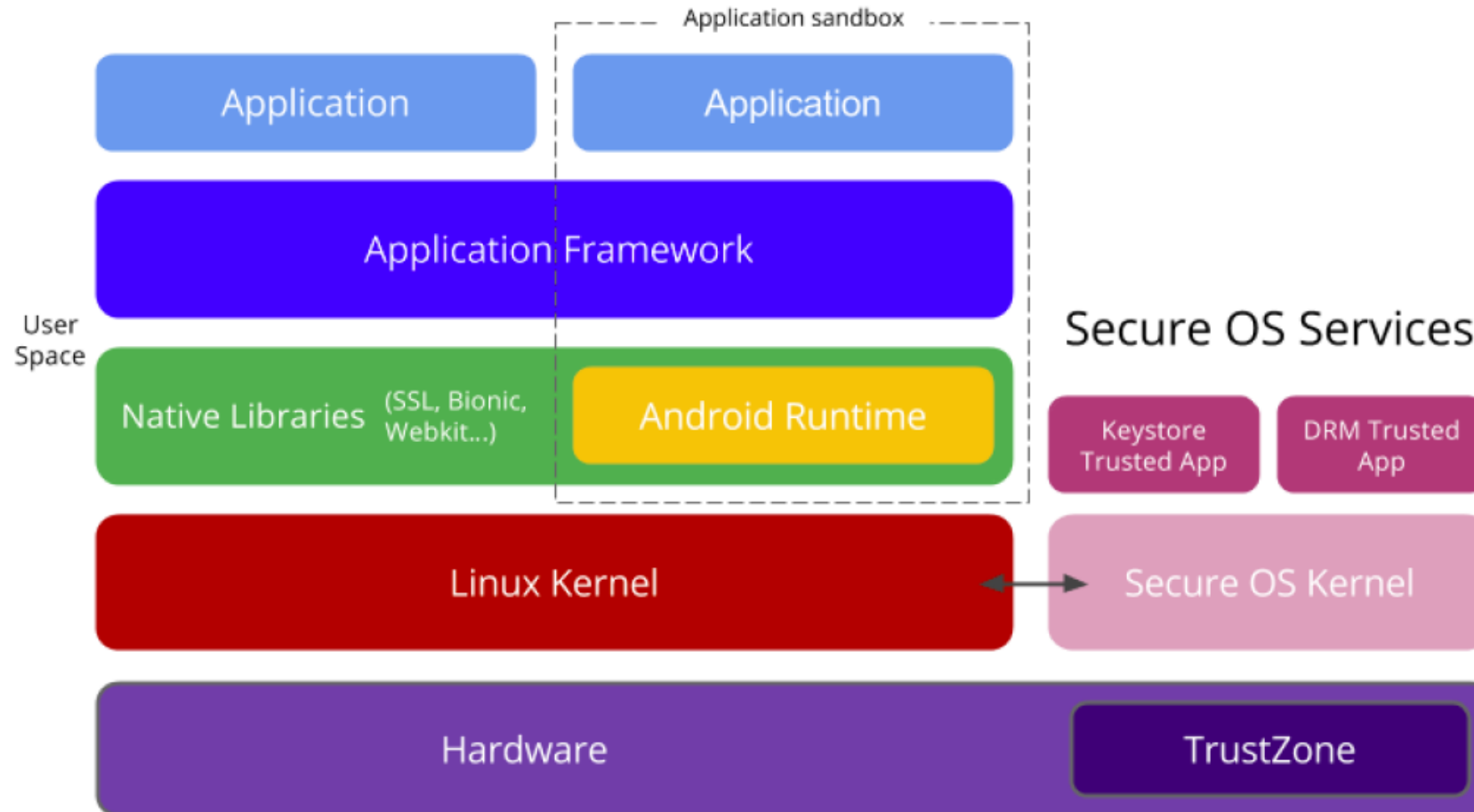
## On Android?

- Device constantly scans for other devices nearby
- Bluetooth implementation runs with privileged permissions and is exploitable (heap overflows)

## Result

- Remote code execution on phone without user noticing

# Android Security Architecture



# **Low-Level System Security**

# Verified Boot



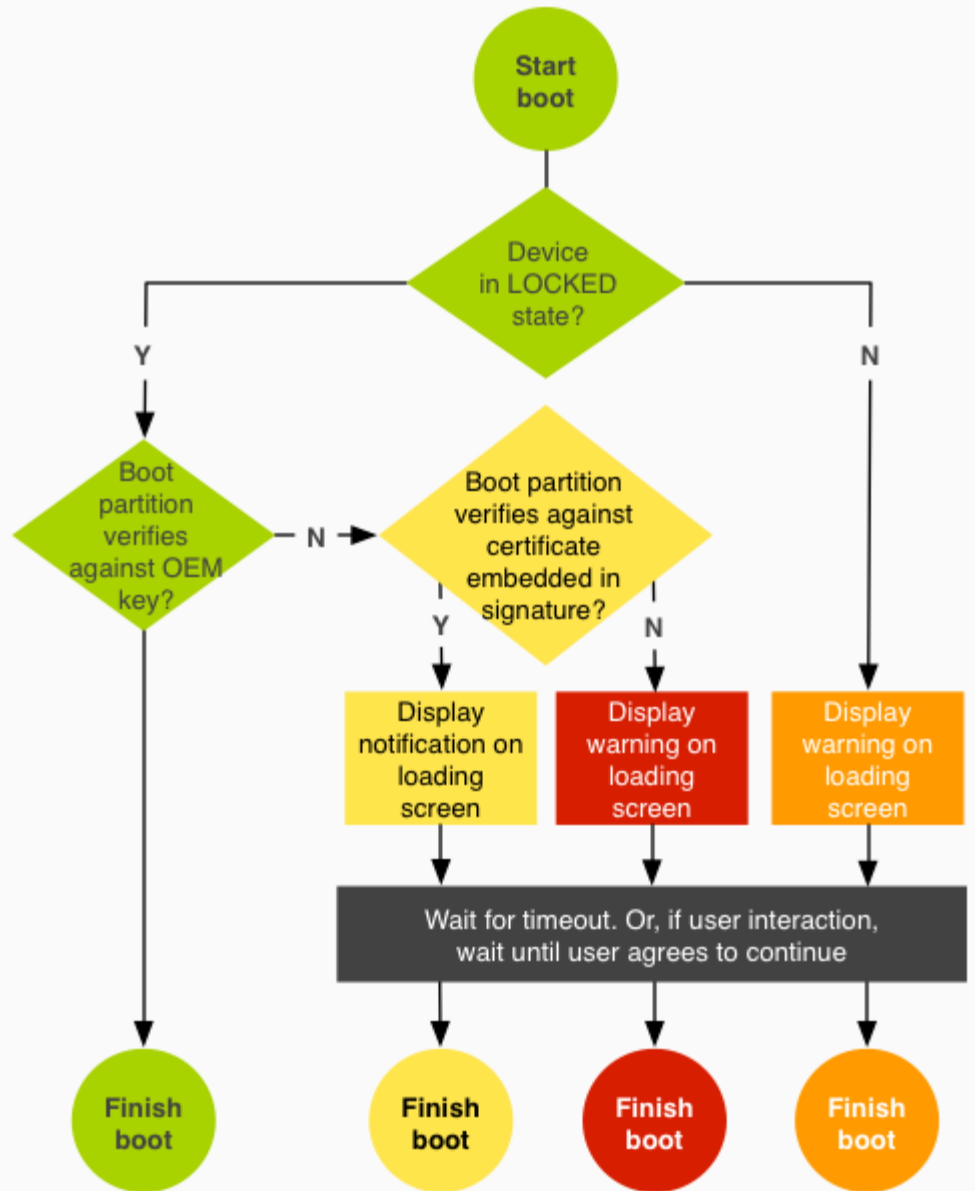
- „Chain of Trust“  
*Established between bootloader and system image*
- Transparent **real-time integrity checking** of block devices  
→ Prevent persistent rootkits
- Based on Device Mapper verity (dm-verity) feature of Linux Kernel  
→ Protection only effective if kernel can be trusted

Typical for OEMs:

Unmodifiable keys *burned* into device to verify boot partition's signature



# Verified Boot – Workflow



## Boot chain (simplified)

- Verify bootloader using Chain of Trust
- Bootloader verifies boot / recovery partition
- Kernel verifies system partition

## Device / bootloader state

- LOCKED/UNLOCKED
- Allows custom (non-OEM) keys

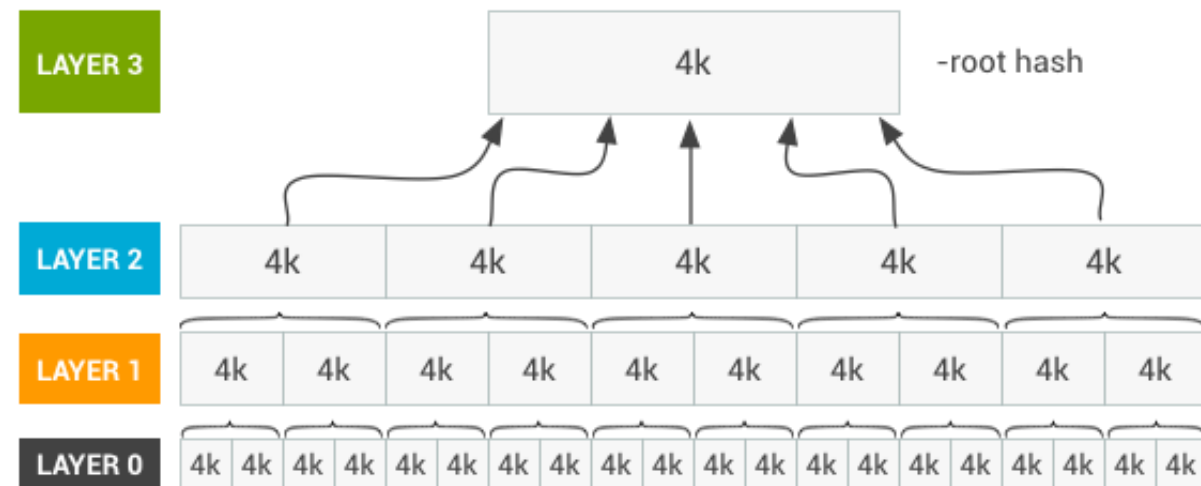
## Boot state

- GREEN/YELLOW/ORANGE/RED
- Does not stop boot, only warning

# dm-verity – Insight

**Idea:** Look at block device and storage layer of file system using a hash tree

- Hash values stored in tree of pages
  - Only „root hash“ must be trusted to verify rest of tree
- Modification of any 4k-block would change the „root hash“
- Verify signature of „root hash“ using public key included on boot partition  
→ Confirm that device’s system partition is unchanged



# dm-verity

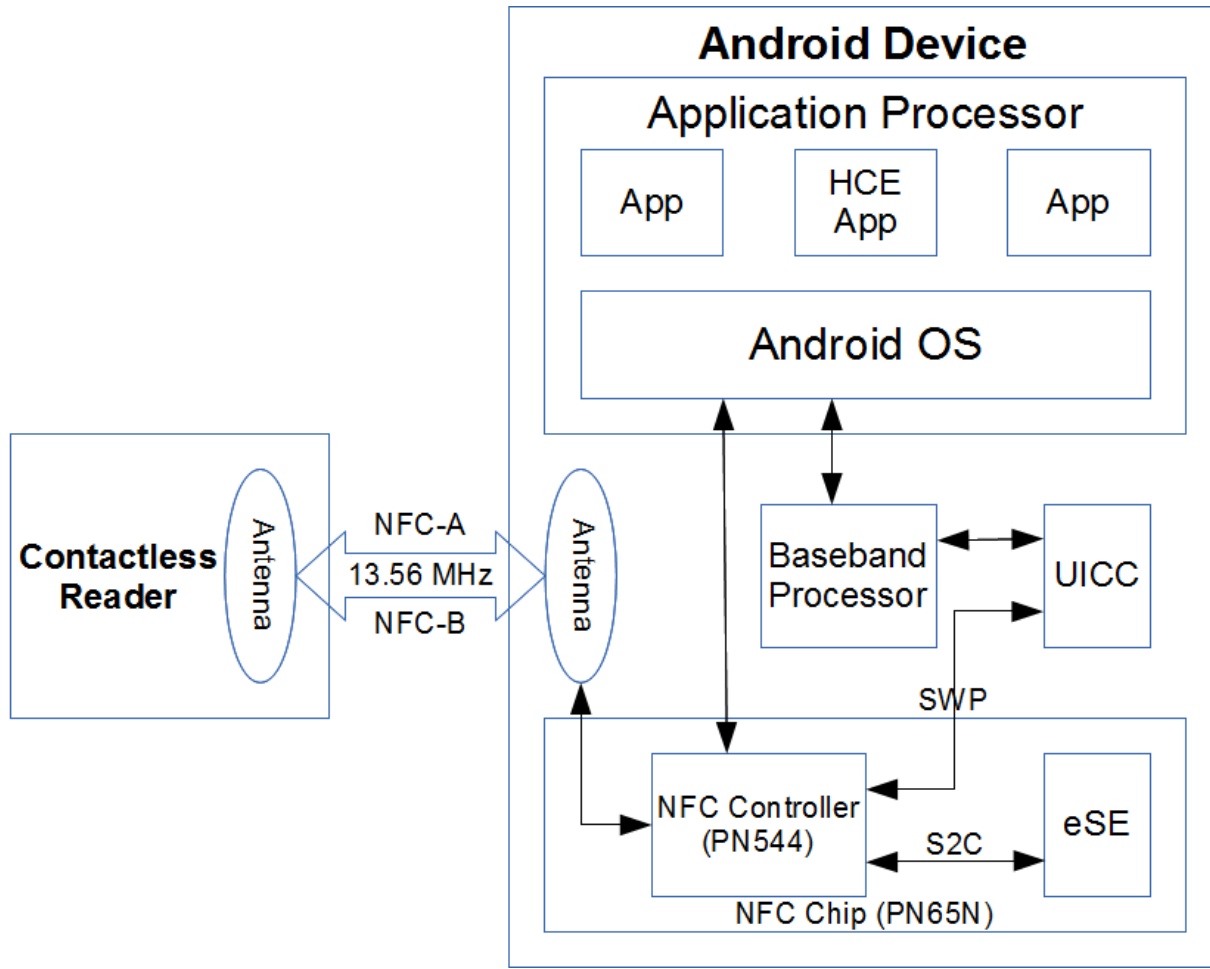
## Limitations

- Only applicable to *read-only* partitions
  - *Read-write* partitions would update metadata when files are read
  - Any change in FS breaks the tree
    - **but useful** for /system partition (or where *read-only* is no drawback)
- Need block-based OTA updates
  - Need to ensure that all devices have same /system partition

## Status on Android 11

- Default is *enforcing* mode, fallback to *logging* mode if metadata unverifiable
- State saved in dedicated metadata partition

# Device Interfaces



## Near Field Communication (NFC)

- Read/write mode (RW)
- Peer-to-peer mode (P2P)
- Card emulation mode (CE)

## Secure Elements

- SIM card (UICC)
- microSD card (ASSD)
- Embedded SE (eSE)

## APIs

- Telephony APIs (restricted)
- Android HCE (HostApduService)
- OpenMobile API (SEEK)

# Encryption System

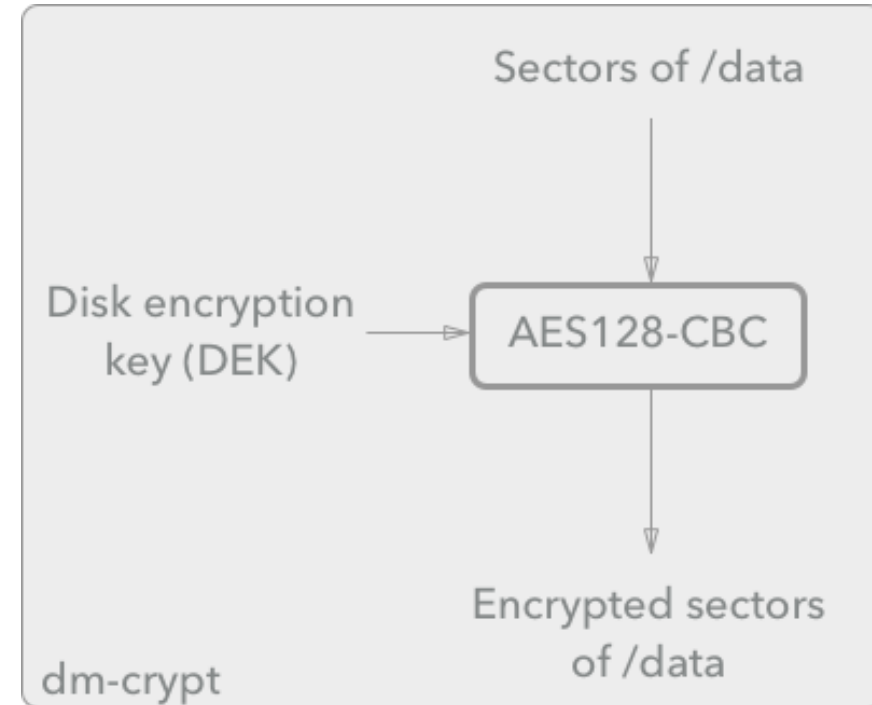
# Overview

## *Starting with Android 3.0...*

- 4.4: Replaced PBKDF2 with scrypt
- 5.0: Hardware-backed key storage
- 7.0: Introduced file-based encryption

## Full Disk Encryption

- Uses dm-crypt
- Operates on block-level
- Random-generated 128-bit disk encryption key (DEK)
  - < 5.0: Key file protected only by lock screen password
  - Now: Key file stored in Secure Element



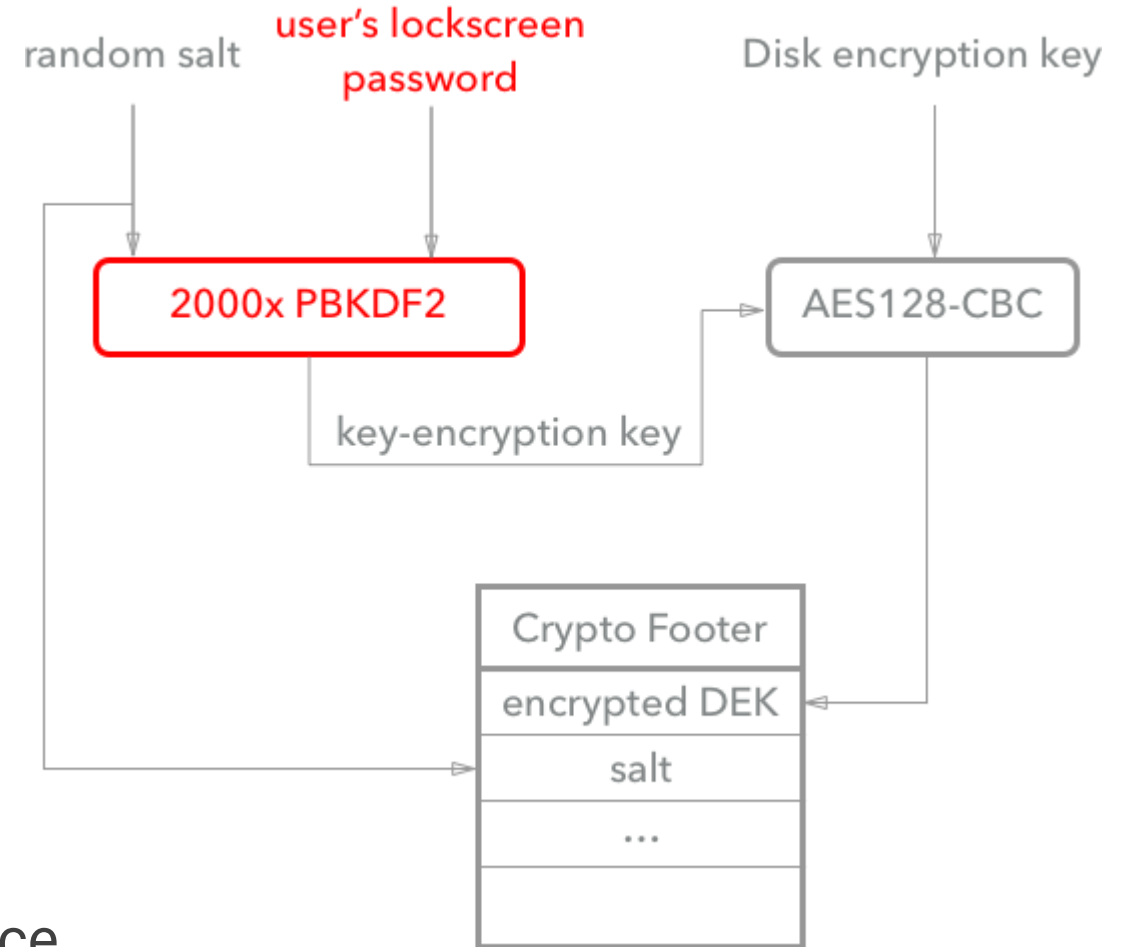
# FDE in Android 3.0

## PBKDF2 with 2000 iterations

- < 16 chars lockscreen password
- Random salt
- Derivation based on SHA-1
  - Needs only little memory
  - Attack parallelizable :-)

## Brute-Force Attack

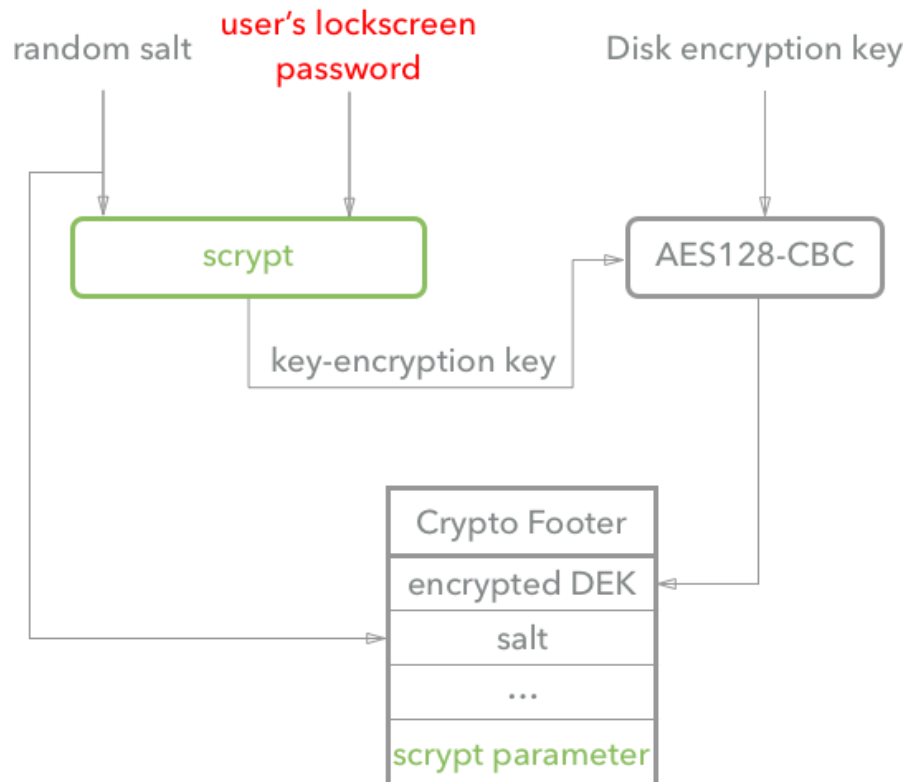
- Copy encrypted /data & crypto footer off device
  - Crypto footer found with „encryptable“ flag in /etc/fstab
- Brute force via GPU, validate key by decrypting → 6-digit PIN needs only seconds!



# FDE in Android 4.4

## Scrypt KDF instead of PBKDF2

- Salsa20-like hash function instead of SHA-1
- Prevent parallelizable large-scale attacks using „work factors



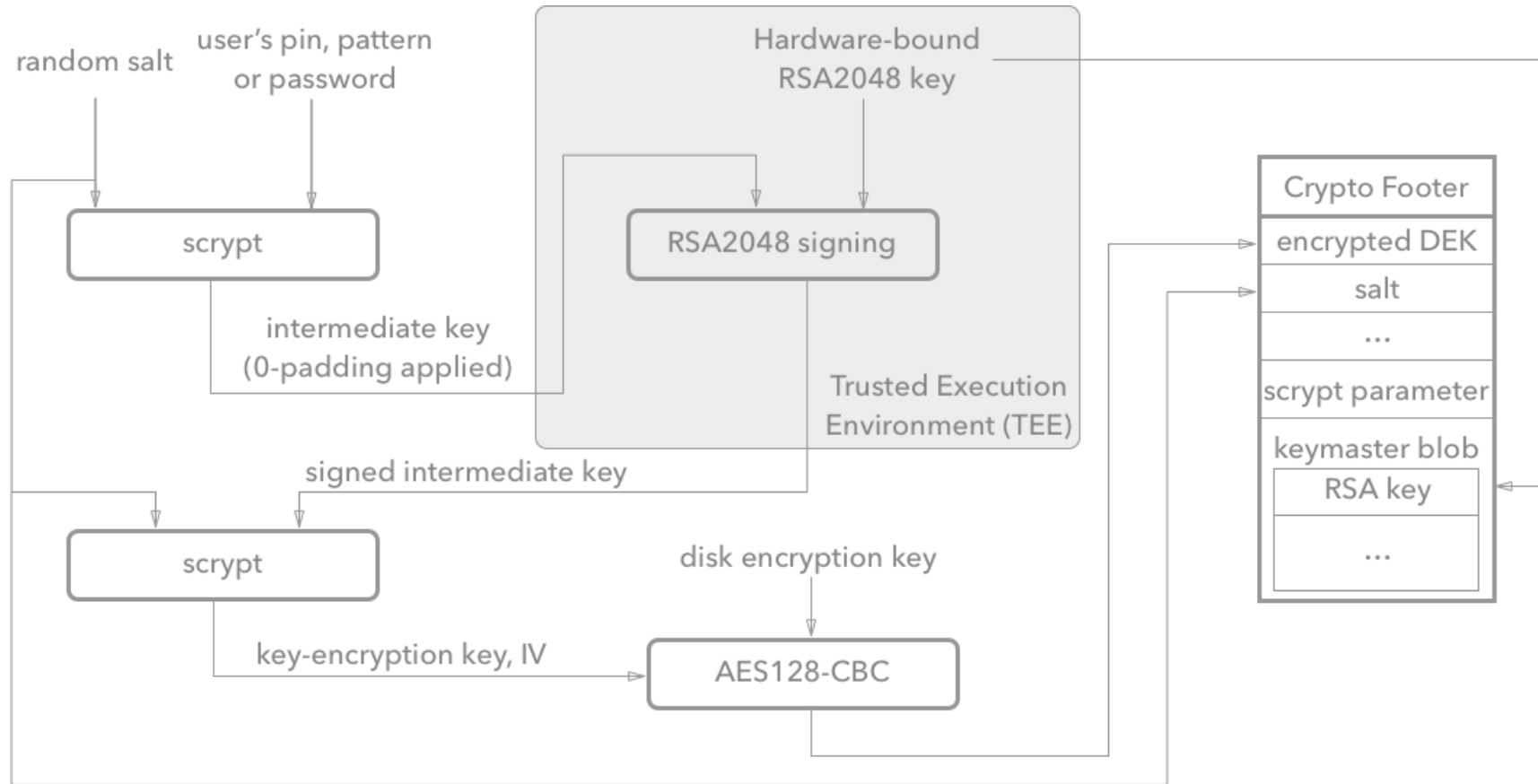
```
$ time python bruteforce_stdcrypto.py header footer 4
Android FDE crypto footer
-----
Magic           : 0xD0B5B1C4
Major Version   : 1
Minor Version   : 2
Footer Size     : 192 bytes
Flags           : 0x00000000
Key Size        : 128 bits
Failed Decrypts: 0
Crypto Type     : aes-cbc-essiv:sha256
Encrypted Key   : 0x66C446E04854202F9F43D69878929C4A
Salt            : 0x3AB4FA74A1D6E87FAFFB74D4BC2D4013
KDF             : scrypt
N_factor        : 15      (N=32768)
r_factor        : 3      (r=8)
p_factor        : 1      (p=2)
-----
Trying to Bruteforce Password... please wait
Trying: 0000
Trying: 0001
Trying: 0002
Trying: 0003
...
Trying: 1233
Trying: 1234
Found PIN!: 1234
```

**Brute-Force still possible  
but takes longer!**

See: <https://goo.gl/a7Qjv1>



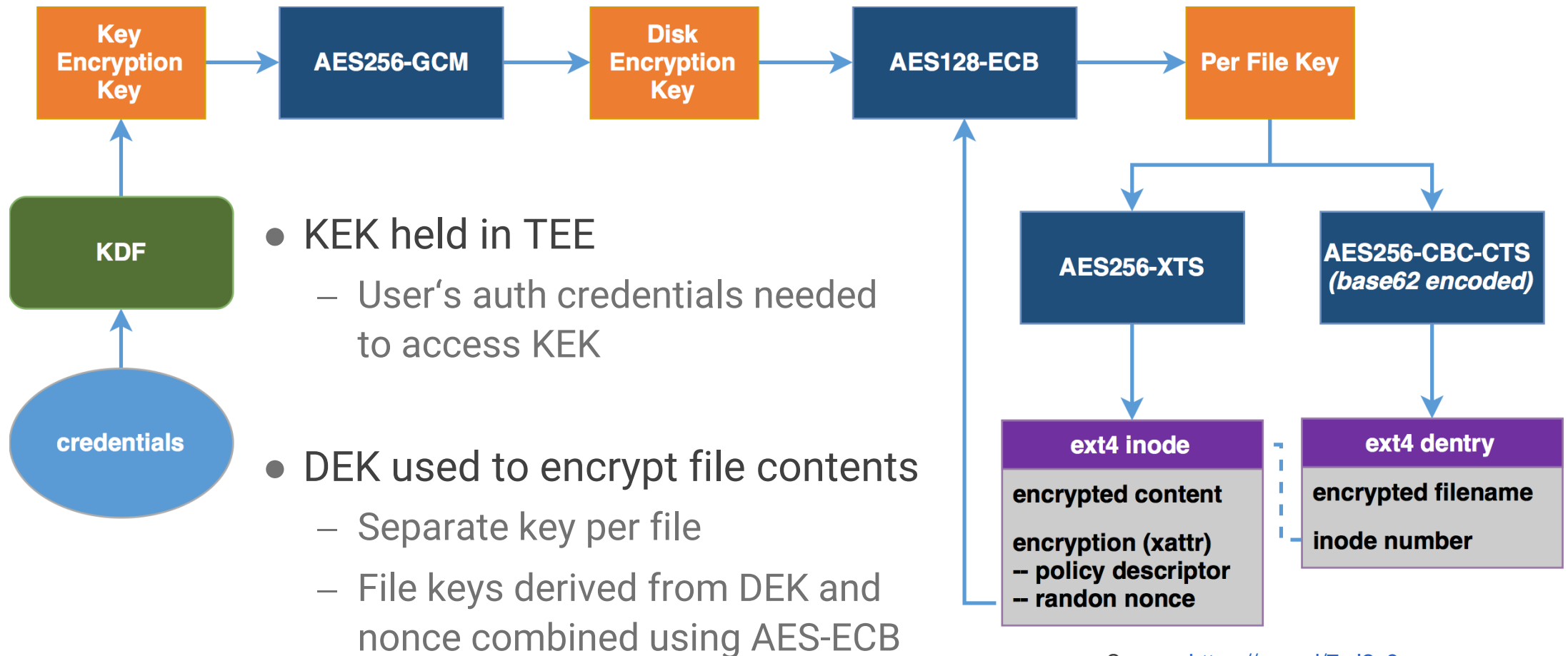
# FDE in Android 5



- Support for patterns and encryption without password
- Hardware-backed key storage for encryption key using signing capabilities of TEE
- „Off device“ brute-force attack no longer feasible

# File-Based Encryption

*Since Android 7.0: Encryption of files instead of block-level*



Source: <https://goo.gl/7zJ2c9>

# File-Based Encryption

Instead of crypto footer for partition,  
key storage in `/data/misc/vold/user_keys`

→ Different subdirectory in ce and de per Android user id

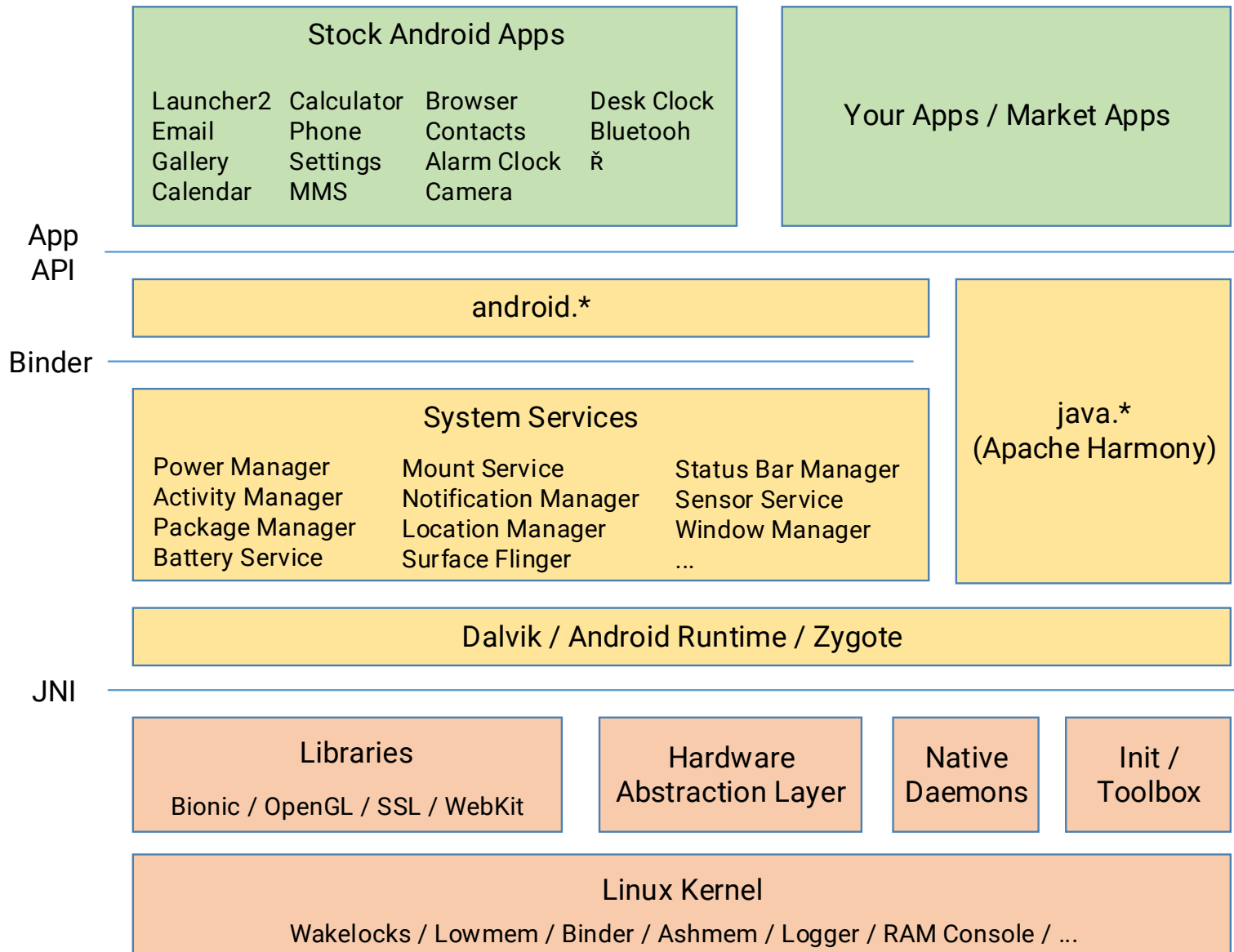
## Two Areas

- Device Encrypted (DE)
  - Immediately available after device turn-on
  - „*Direct boot*“ mode: Receive phone calls, set alarms, ...
- Credential Encrypted (CE)
  - Available after user entered authentication credentials

```
$ ls -R /data/misc/vold/user_keys
+ ce/0/current:
    - encrypted_key
    - keymaster_key_blob
    - salt
    - secdiscardable
    - stretching
    - version
+ de/0:
```

# Android OS Security

# OS Architecture



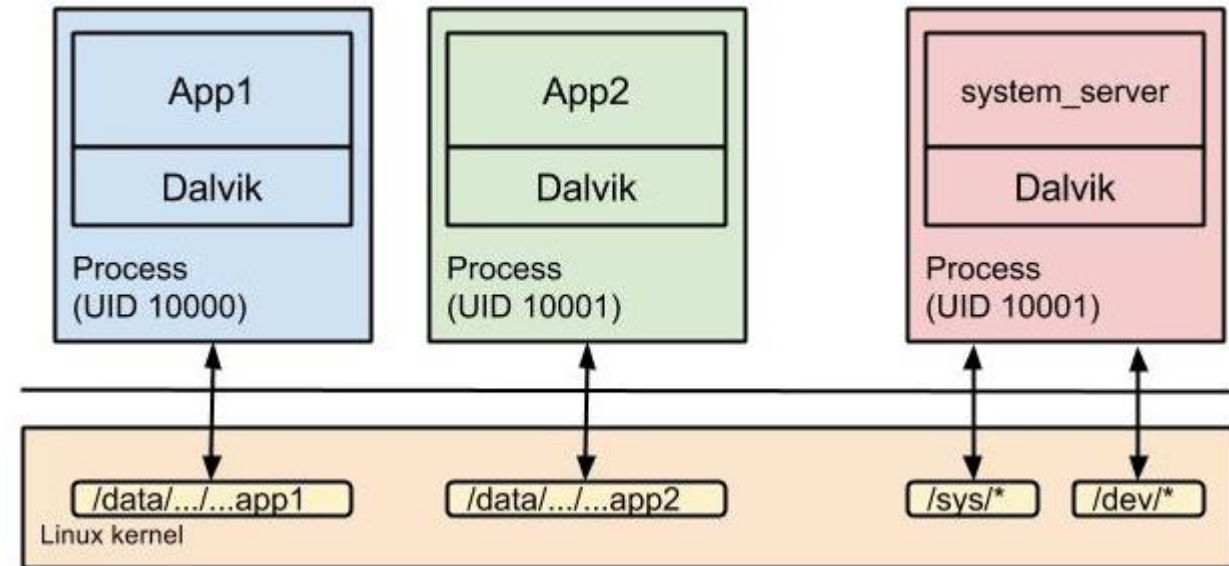
# Android Security Model

- 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)
- Systems running with reduced privileges
- Code signing
  - Application packages (APKs)
  - OS update packages (OTA packages)
- Permissions: System and custom (per app)



# App Sandbox

- Android assigns unique UID to each application → separate processes  
→ Kernel-level application sandbox
- Security enforced at process level through standard Linux facilities (UID, GID)
- Sandbox at kernel level  
→ Security model extends to native code and OS applications too
- FS permissions as a mechanism to keep files / folder separate



# App Sandbox

- **Installing new apps**

- Creates new directory /data/data/<Package name>/
  - E.g. /data/data/com.whatsapp/

```
$ ls -l /data/data/
drwx-----  4 u0_a97          u0_a97          4096 2017-01-18 14:27 com.android.calendar
drwx-----  6 u0_a120         u0_a120         4096 2017-01-19 12:54 com.android.chrome
...
```

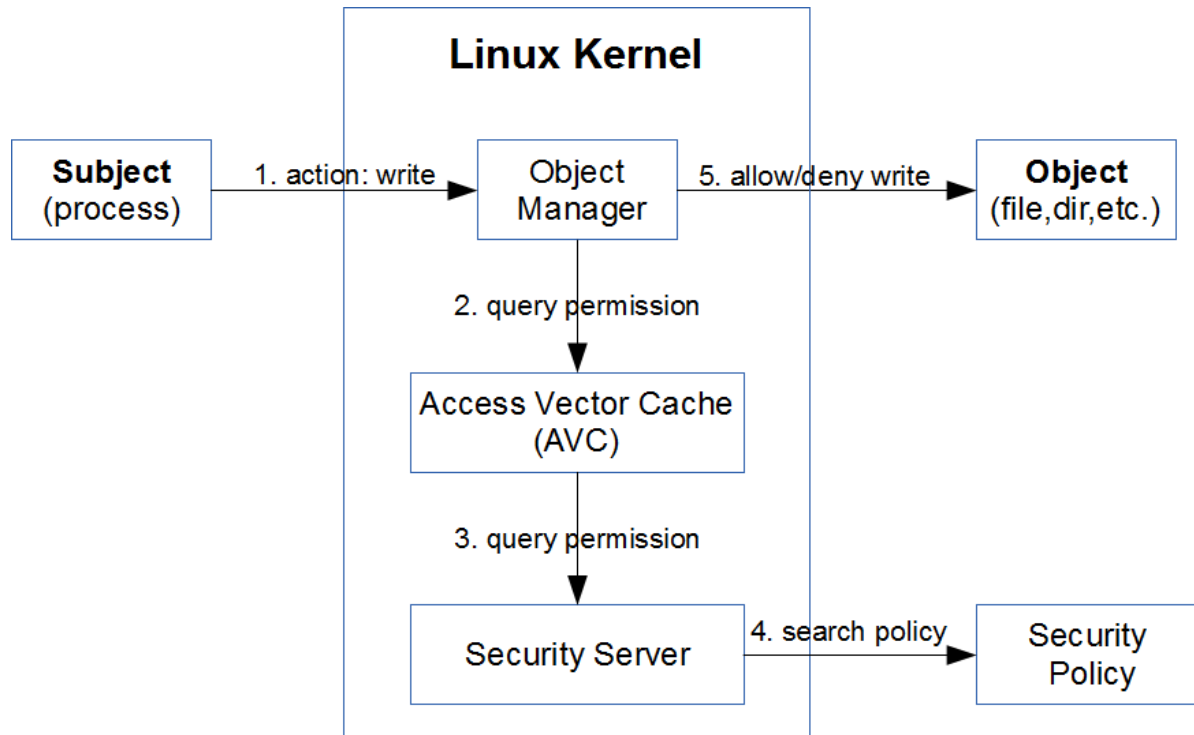
- Accessing other apps' directory → needs same UID
  - Apps signed with same developer certificate
  - Explicitly sharing same UID in AndroidManifest.xml

```
1 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
2   package="com.android.nfc"
3   android:sharedUserId="android.uid.nfc">
```



# SELinux on Android

Security-Enhanced Linux



By default since Android > 4.3:  
Define app boundaries with SELinux

## Concept

„Not explicitly allowed? Then deny!“

## Modes

- Permissive: Denials only logged
- Enforcing: Logged and enforced

Since Android 5: Enforce always (only)

# SELinux on Android – Sample Rules

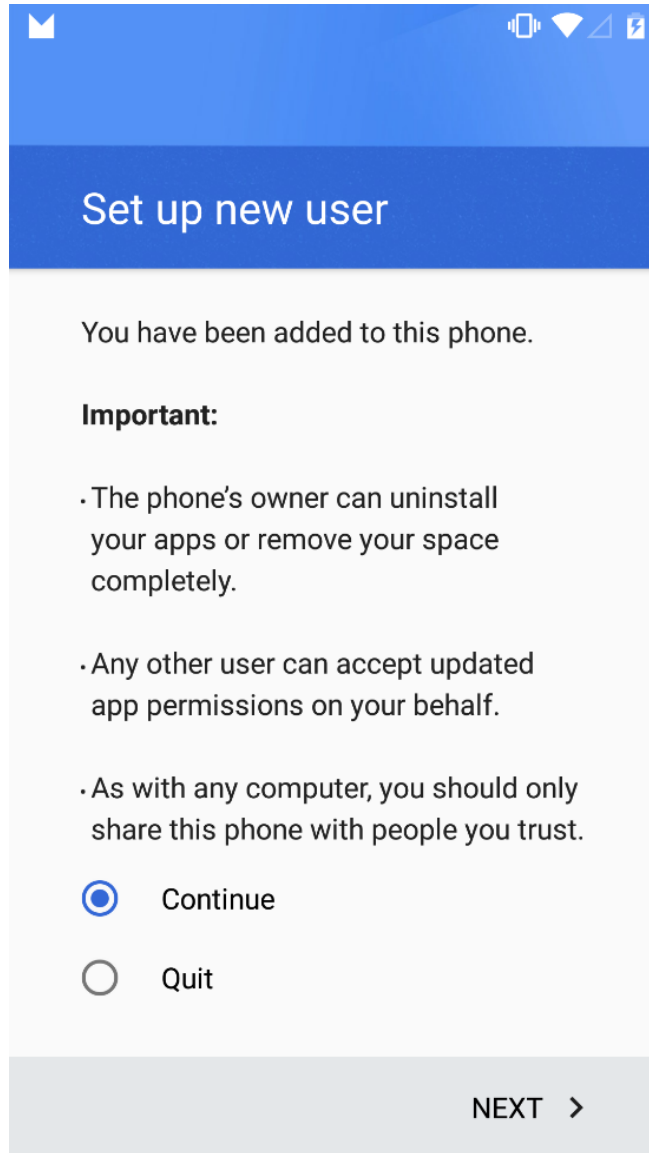
- No unlabeled files
- No ptrace
- No device node creation
- No raw I/O
- No mmap zero
- No mac\_override
- No setting security properties
- No access to /data/security and /data/misc/keystore
- No /dev/mem or /dev/kmem access
- No /proc usermode helpers
- No ptrace of init
- No access to generically labeled /dev/block files
- Restrictions on mounting filesystems
- No execute of files from outside of /system
- No access to /data/properties
- No writing to /system or rootfs
- No registering of unknown services
- No entering init domain
- No /sys/kernel/debug read access
- No apps acquiring capabilities
- No raw app access to camera, microphone, NFC, radio, etc.
- No app-generic socket access
- No app/proc access to different security domains
- No access to GPS files
- Cannot disable SELinux

Meanwhile > 250 Rules

# Multi-User support

- Originally for tablets only, now for phones too (> Android 5.0)
- Users isolated by UID / GID
- Separate settings & 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:  $uid = userId * 10000 + (appId \% 10000)$
  - Shared Apps: Install state in per-user `package-restrictions.xml`
- External storage isolation

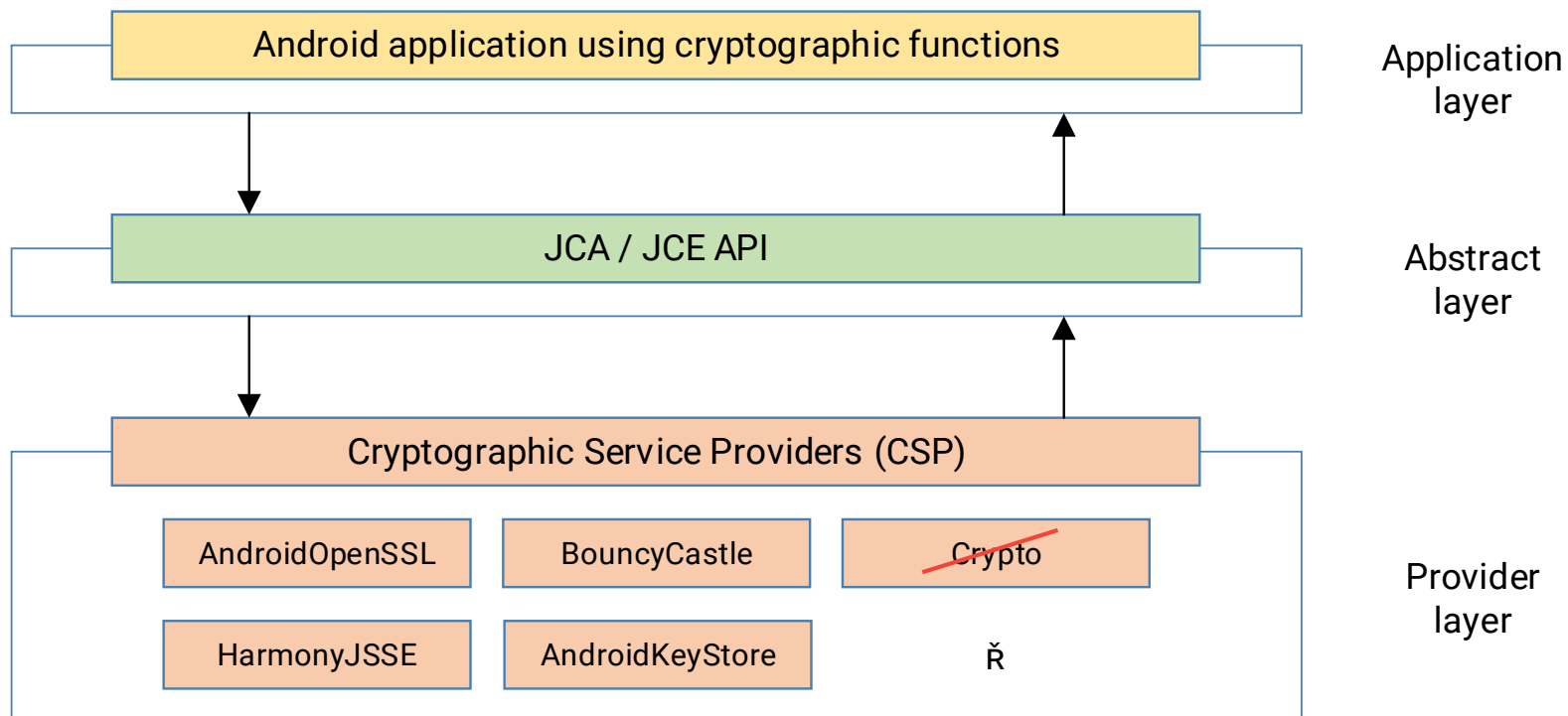
# User Types



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

# Cryptography

- JCA Provider Architecture
- (SSLv3), TLS v1.0-v1.3 support via JSSE API



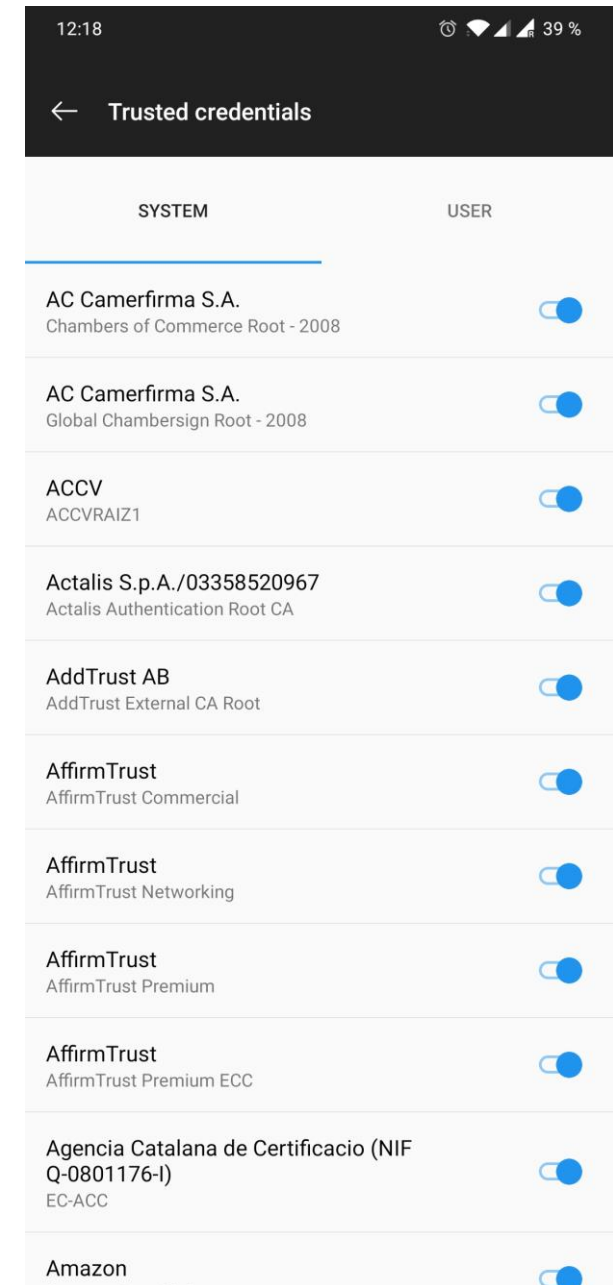
# Cryptography

## *What makes correct Crypto difficult on Android?*

- Insecure defaults imported from Java
  - E.g. Cipher.getInstance(„AES“) implicitly uses ECB mode
  - Bad / no documentation on how to use correctly
- Variety of crypto providers
  - Many apps bundle SpongyCastle library to fix issues in BouncyCastle
  - No full BouncyCastle library in Android → features depending on included version
- Frequent changes in APIs
  - **Android 7:** „Crypto“ provider deprecated, SHA1PRNG replaced with OpenSSLRandom
  - **Android 8:** „You should not use IVParameterSpec for GCM but GCMParameterClass“
  - **Android 9:** „Crypto“ provider removed, developer must not explicitly select provider

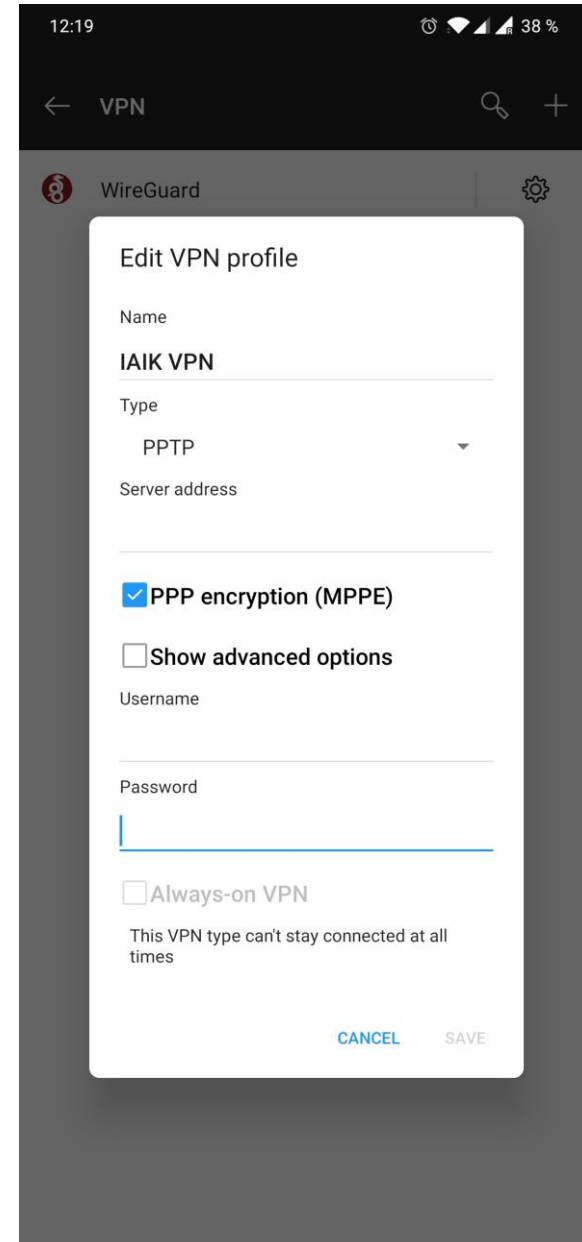
# Certificates & PKI

- Android-specific trust store
- Trust anchors
  - Pre-installed („trusted credentials“)
  - Per user / profile
- Modified certificate building chain
  - Based on BouncyCastle code
  - Dynamically updated certificate blacklists
  - Dynamically updated Certificate Pinning for Google Sites



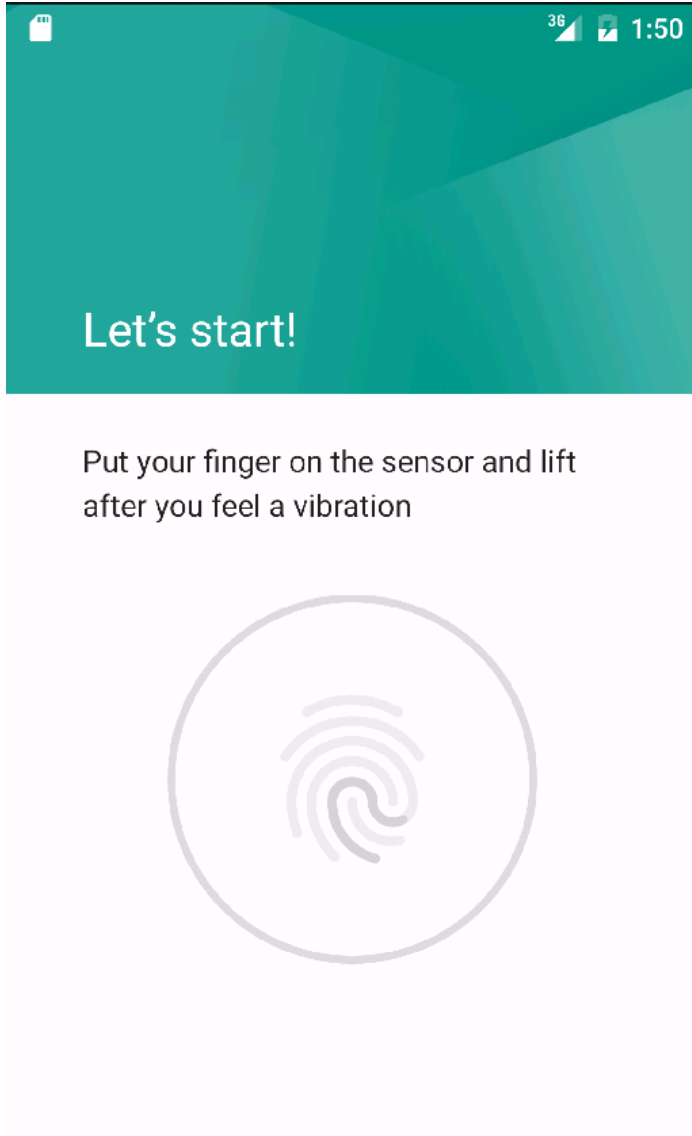
# Networks

- WPA EAP2 Enterprise (802.11i)
  - EAP: EAP-TLS, EAP-TTLS, PEAP, EAP-SIM, EAP-AKA since Android 5.0
  - Integrates with system keystore
  - Integrated with Android for Work (device administrator APIs)
- VPN
  - Legacy: PPTP, 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





# Device Security

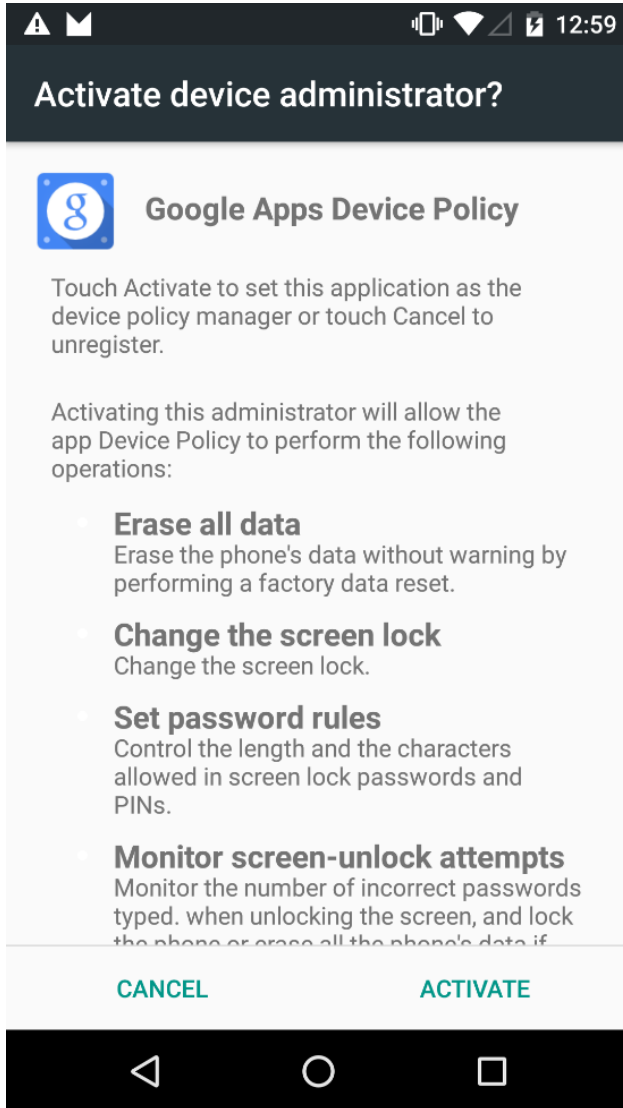


- Lockscreen (keyguard service)
  - Pattern (least secure)
  - PIN / Password
  - Stores hashes, uses Gatekeeper HAL since Android 6.0
- Smart Lock since Android 5.0
  - Extensible Trust Agents
  - Bluetooth, NFC, Location, Face Recognition
- Factory reset protection since Android 5.1
  - Google account info saved on frp partition
- Fingerprint API since Android 6.0

# Credential Storage

- System-managed, secure cryptographic key store
  - Unexportable keys
  - Remain secure even if OS is compromised → Secure Element
- Implemented in the keystore system service
  - HAL interface (keymaster), hardware-backed implementations possible
  - Typically uses TEE (implemented using TrustZone) on ARM devices
- Framework APIs
  - KeyChain API
  - KeyStore
  - KeyPairGenerator, KeyGenerator

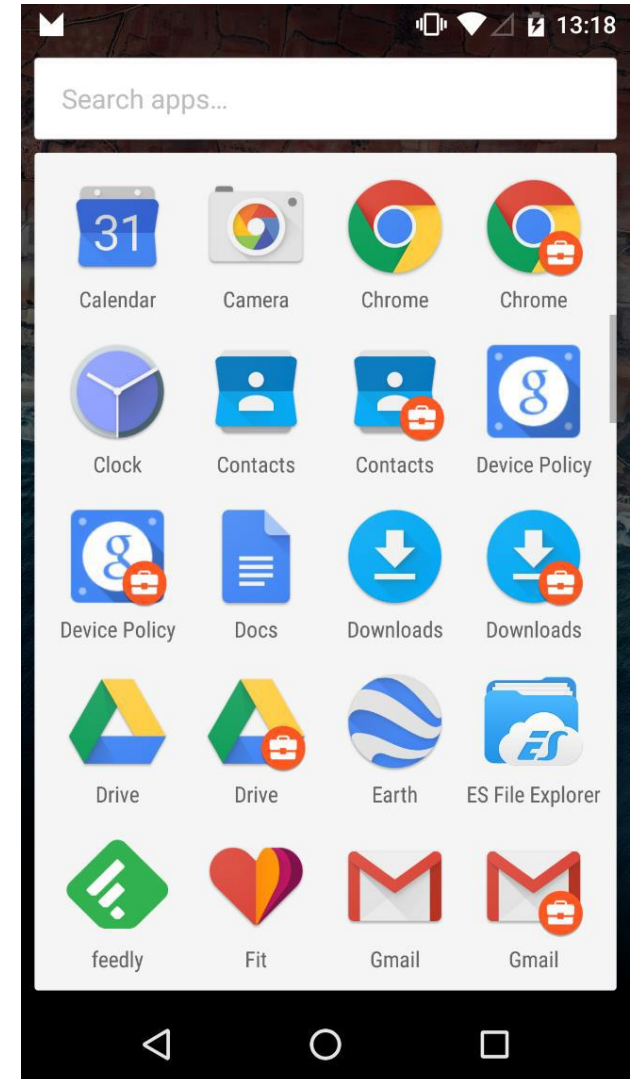
# MDM



- Device security policy can be set by admin
  - Password / PIN policy
  - Device lock / unlock
  - Storage encryption
  - Camera access
- Needs to be activated by user
- Cannot be directly uninstalled
- May be required to sync account data
  - Microsoft Exchange (EAS)
  - Google Apps

# Android for Work

- Android > 5.0 provides “Work Profiles”
  - Pre-defined managed provisioning flow
  - Managed by “Profile Owner” (device admin)
  - Requires device encryption
- Separate apps and data: Can only install approved apps
- UI shared with primary user (Launcher, Notifications, ...)
- “Device owner” is super-device admin
  - Installed upon first device initialization
  - Cannot be uninstalled
  - Extra privileges
  - Scoped to whole device



# Outlook

- 29.04.2021
  - Application Security on Android
  
- 06.05.2021
  - Static and Dynamic Application Analysis

