The Android Platform Security Model
(and the security status of actual devices)

Cambridge University Mobile System Group, 2020-11-30 14:00 (UTC+0), virtual

Univ.-Prof. Dr. René Mayrhofer (JKU Linz)

(Full disclosure: also affiliated with Android security, but not speaking for Google today)

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Straße 69
4040 Linz, Austria
jku.at

Context: Convergence of security-critical services

Context: The Android ecosystem

… is massive, diverse, and constantly changing

- >1.300 brands
- >24.000 devices
- >1.000.000 apps
- >2.000.000.000 users

(https://www.blog.google/around-the-globe/google-europe/android-has-created-more-choice-not-less/)

Context: The Android ecosystem

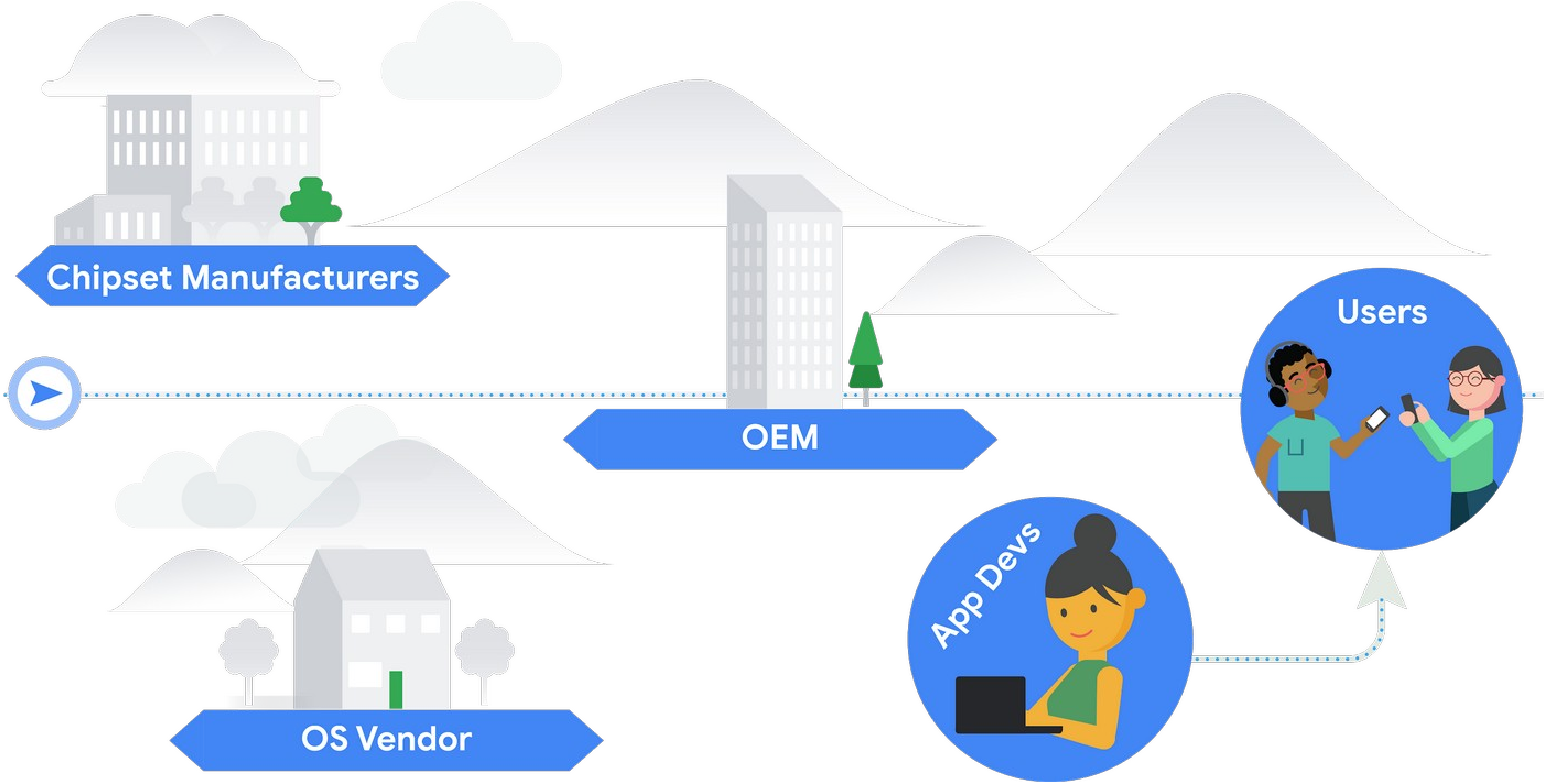


Image credit: Google

Context: The Android ecosystem

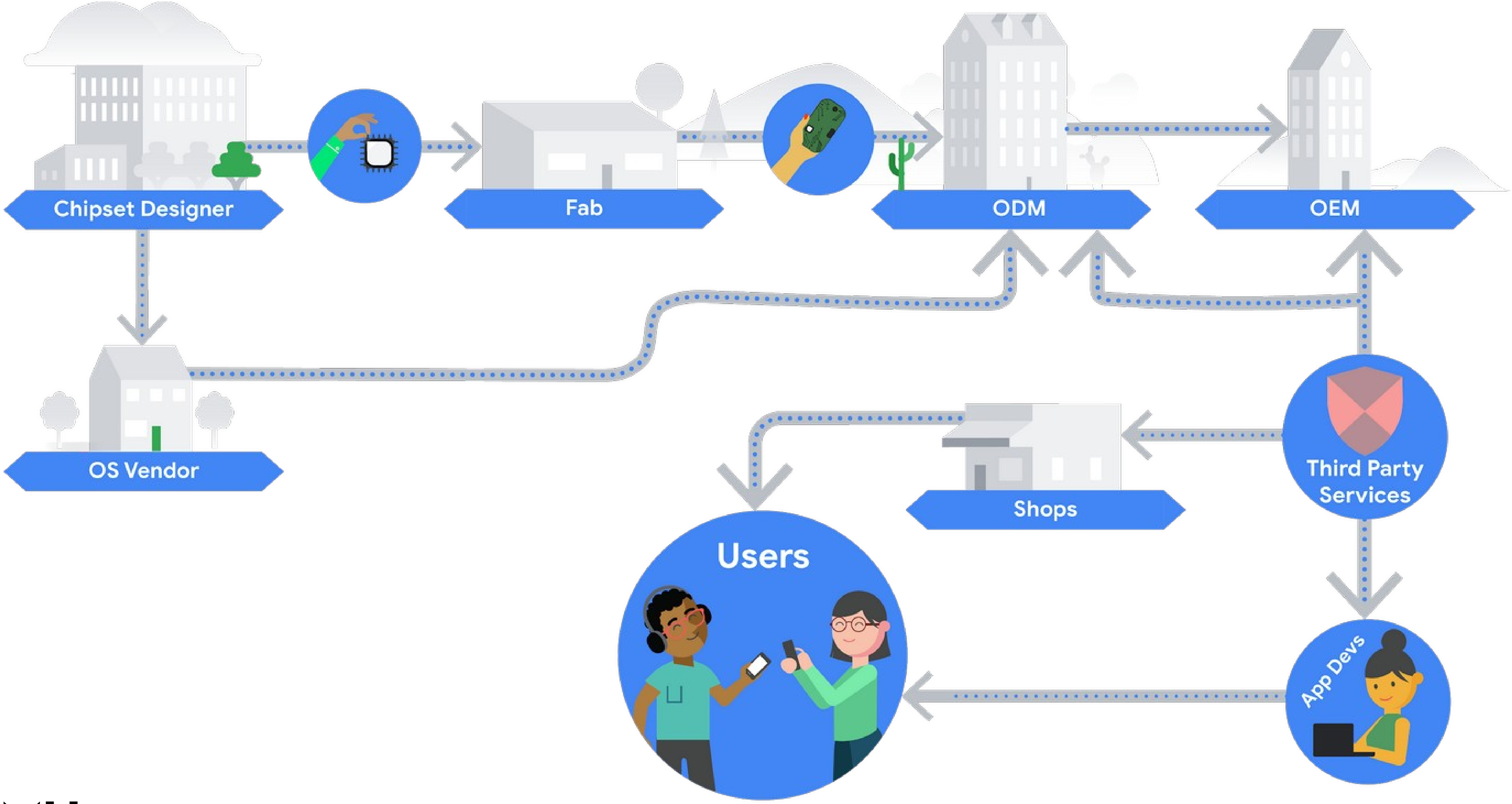


Image credit: Google

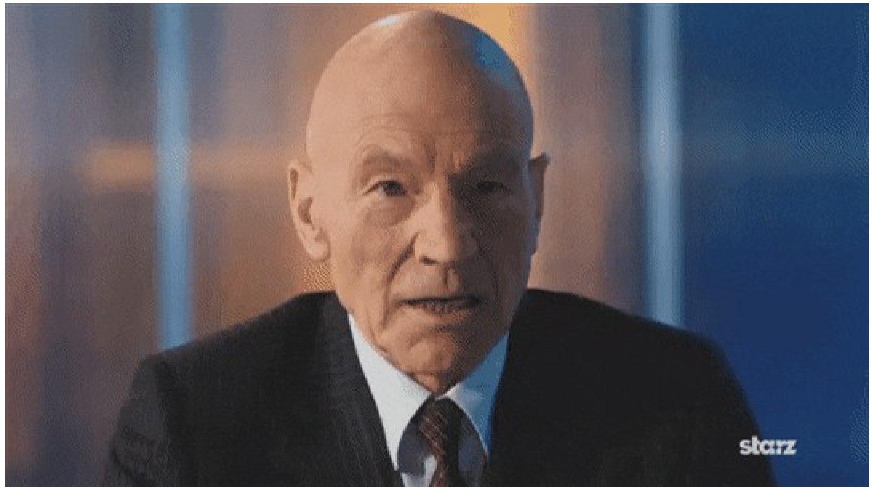
The Android Platform Security Model: Security Goals

1) Protecting **user data**
   - Usual: device encryption, user authentication, memory/process isolation
   - Upcoming: personalized ML on device
2) Protecting **device integrity**
   - Usual: malicious modification of devices
   - Interesting question: against whom?
3) Protecting **developer data**
   - Content
   - IP

[R. Mayrhofer, J. Vander Stoep, C. Brubaker, N. Kralevich. “The Android Platform Security Model‘, arXiv:1904.05572, April 2019]

The Android Platform Security Model: Threat Model

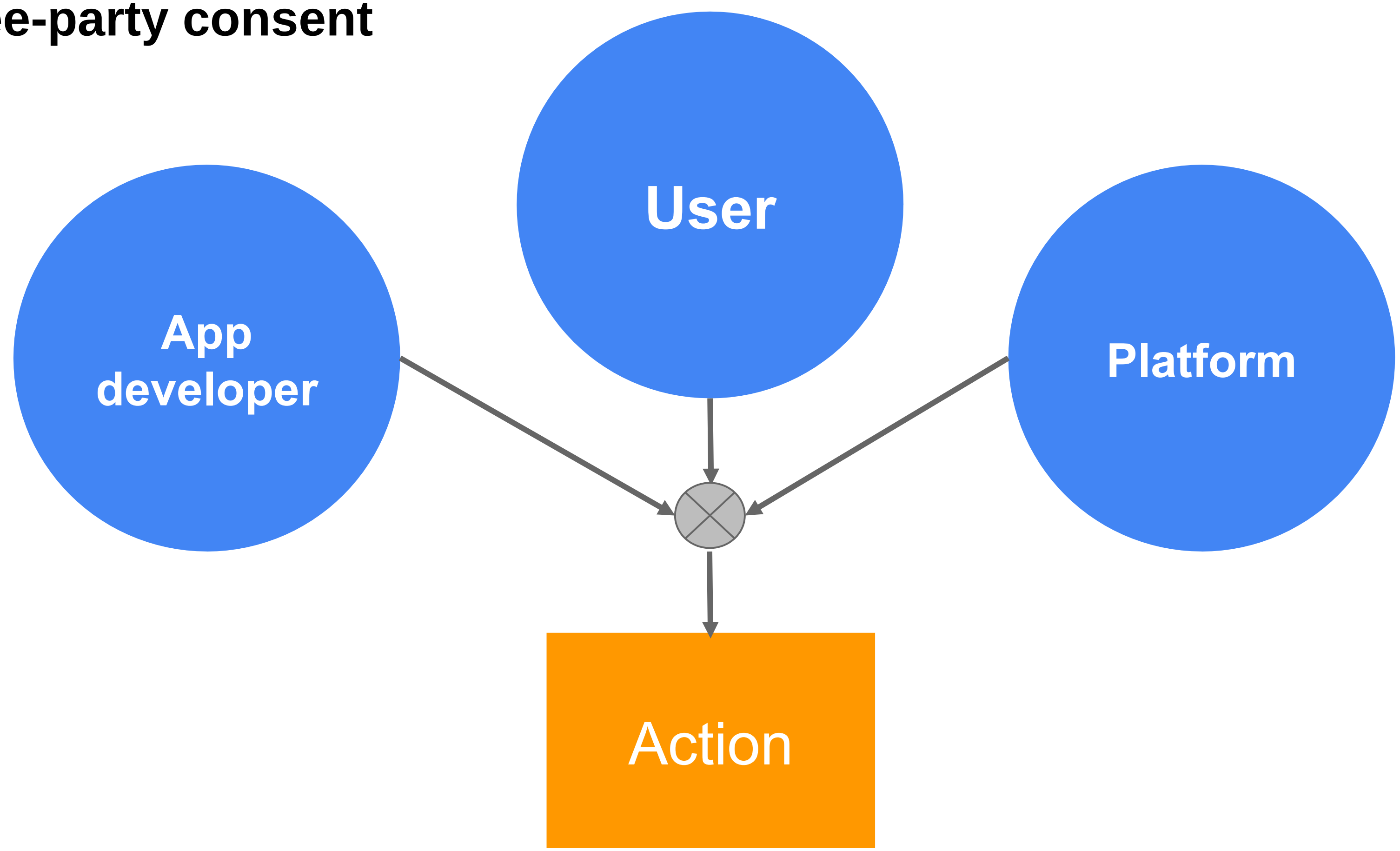
- Adversaries can get **physical access** to Android devices (lost, stolen, borrowed, etc.)
  - Physical proximity
  - Powered off
  - Screen locked
  - Screen unlocked by different user
- **Network communication** and **sensor data** are untrusted
  - Passive eavesdropping
  - Active On-Path Attacker (OPA) / MITM
- **Untrusted code** is executed on the device
  - Includes all forms of OS/app API abuse
  - Includes misdirection, deception, etc. through UI
- **Untrusted content** is processed by the device
- New: Insiders can get access to signing keys



[R. Mayrhofer, J. Vander Stoep, C. Brubaker, N. Kralevich. “The Android Platform Security Model‘, arXiv:1904.05572, April 2019]

The Android Platform Security Model: Rules

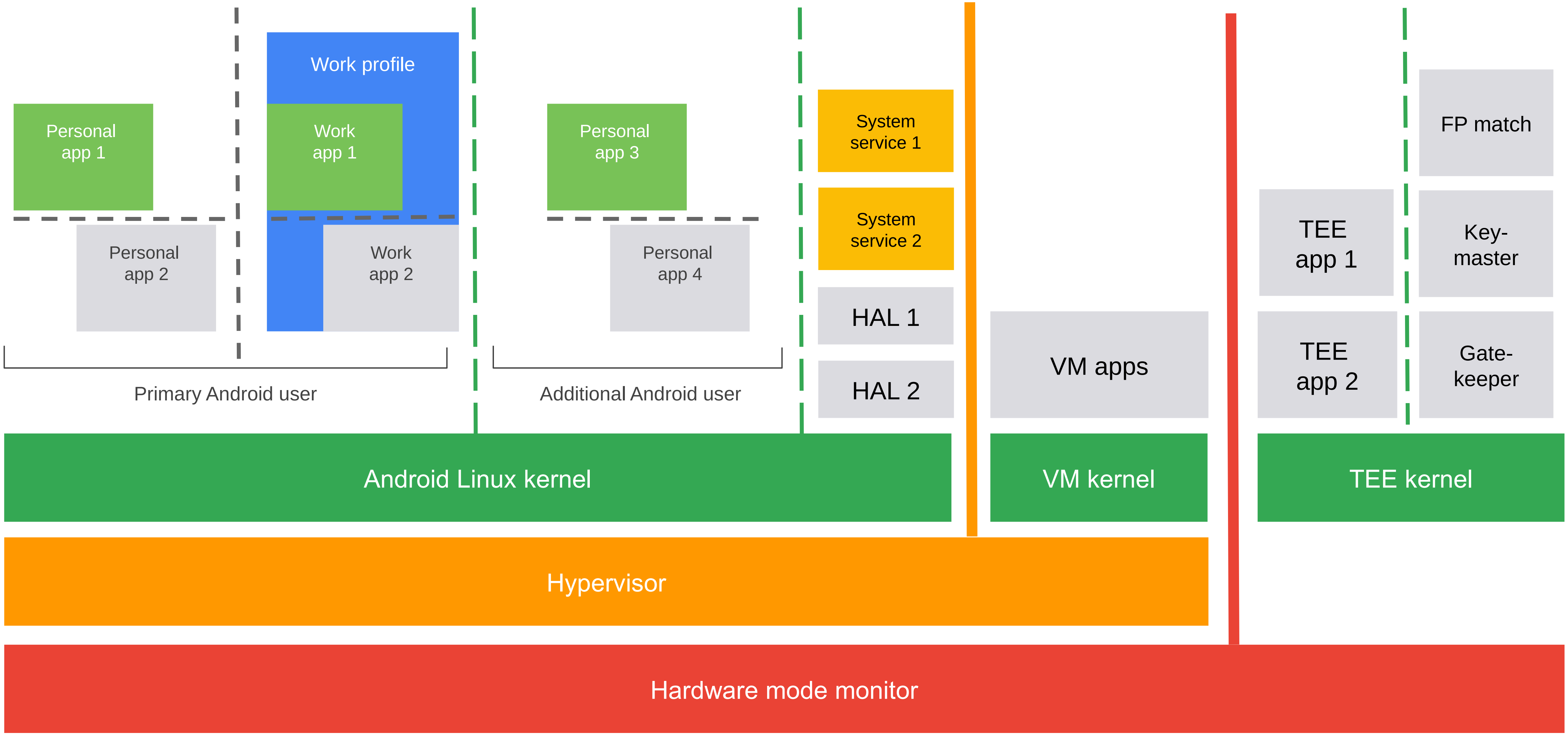
- Rule 1: **Three-party consent**



The Android Platform Security Model: Rules

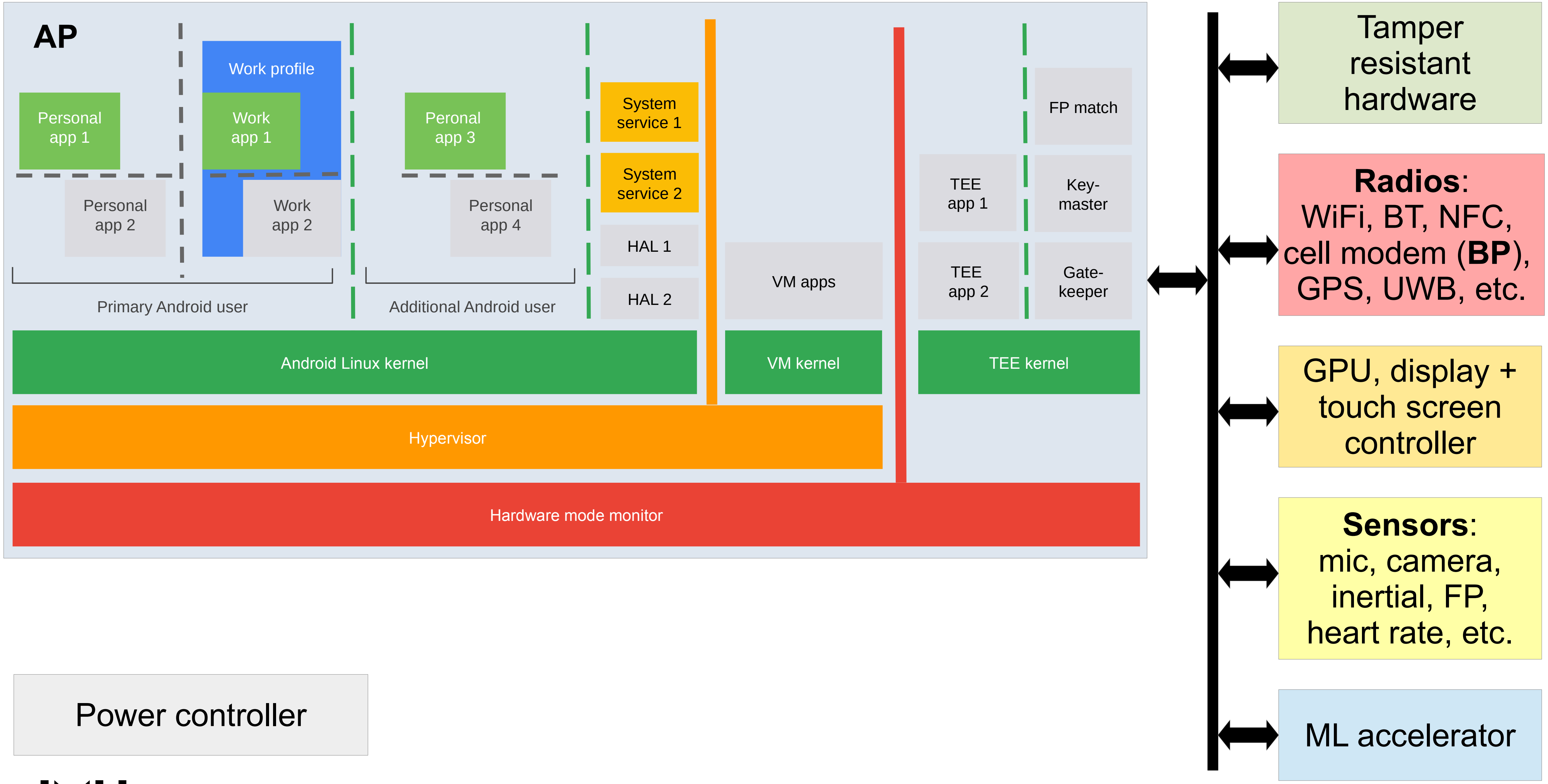
- Rule 2: **Open ecosystem access**
- Rule 3: **Security is a compatibility requirement**
- Rule 4: **Factory reset restores the device to a safe state**
- Rule 5: **Applications are security principals**

Android architecture: layers of isolation (on main CPU)



Question:
Controlling device-wide parameters from work profile?

Android architecture: isolation between hardware modules



Android app security principles

- **Applications must be signed for installation**
  - May be self-signed by the developer, therefore no requirement for centralized application Q/A or control
    - Note: Play-signed apps hold their private signing keys on the Google Play store
  - Signature supports non-repudiability (if the public key/certificate is known)
  - Signature by same private key allows applications to share data and files
  - Automatic application updates possible when signed by same private key

- **Otherwise, open eco-system**
  - Users may install arbitrary applications (directly from APK files or from different markets)
  - **Apps can be written in any language**
  - DRM and application copy protection available (Android 2.2 and newer market API), but optional

Android security architecture

Upon installation, package manager creates a dynamic user ID for each application
⇒ Application sandbox

- All application files and processes are restricted to this UID
- Enforced by Linux kernel and therefore same restrictions for all code (Java + native)
- Starting with Android 4.4 (introduced in 4.3 with `permissive` mode, 4.4 switches to `enforcing`), augmented with **SELinux** policy for kernel level mandatory access control (MAC)
- By default, even the user and debugging shells are restricted to a special UID (`SHELL`)
- Permissions granted at installation time allow to call services outside the application sandbox

"rooting" to gain "root" access (super user / system level access on UNIX without further restrictions, but may be limited by SELinux MAC)

Android security boundaries

Android sandbox has **two main layers of permissions models**

- **File system entries and some other kernel resources**
  - enforced by DAC (standard filesystem permissions) and in newer versions MAC (SELinux) ⇒ enforced on kernel level
  - very restrictive compared to standard Linux distributions
  - Android ID (AID) is used as both UID (user ID, for installed applications) and GID (group ID, for accessing resources)
  - commonly referred to with the term “Android sandbox” (although this is not the full picture)
- **Permissions on API calls**
  - enforced by DalvikVM/ART and Android framework/libraries, as well as specific apps
  - allow bridging the security boundary created by the first layer enforced by kernel sandbox
- Plus other mechanisms for specific purpose (e.g. Linux capabilities and `seccomp` filters)

For interplay between DAC, MAC, and CAP see e.g. [Hernandez et al.: “*BigMAC: Fine-Grained Policy Analysis of Android Firmware*”, USENIX Security 2020], online at https://www.usenix.org/conference/usenixsecurity20/presentation/hernandez

Crossing the app sandbox (process) boundary

- Apps invoke Android APIs as libraries linked in their own process (with the app AID)
- Privileged processes (services) run in different process (other, more privileged AID)
- Crossing the boundary required IPC (Inter Process Communication)
- On Android, implemented by **Binder**
  - Patch to Linux kernel, part of the Android Common Kernel
  - Can be called from unprivileged processes
  - Calls registered objects in other processes
  - Transports objects (shared memory) from one process to another
  - Object-oriented call and arguments interface defined by AIDL (Android Interface Definition Language) ⇒ Details see https://developer.android.com/guide/components/aidl
- **One of the core security components in AOSP** ⇒ bugs in Binder often lead to universal Android exploits

Android code signing

- **All Android apps (system and user-installed) must be signed**
  - typically, firmware updates are also signed by OEM, boot loader may only allow to flash and/or boot "correctly" signed images
  - recovery mode often applies only updates signed by same OEM
  - newer Android versions **verify signatures during boot and run-time** (*dm-verity*)
- Signing is done with private keys held by developers / organizations, public keys embedded in individual apps, system image, and/or in boot loader for image signatures
- Signing key types:
  - individual developer keys (self-signed) for apps
  - `platform`, `shared`, `media` and `testkey` in AOSP tree
    - `platform` is used for "core" Android components with elevated privileges
  - `releasekey` for release type image builds, must by kept private
  - more details at https://source.android.com/devices/tech/ota/sign_builds.html and http://nelenkov.blogspot.co.at/2013/05/code-signing-in-androids-security-model.html

Question:
Make verified boot state available to all apps?`

Taming complexity in variants

Compatibility Definition Document
(Standards)

- Defines requirements a device needs to fulfill to be considered "Android"
- Updated for every Android release
  - Many changes scoped to apps targeting this version
- Needs to strike balance between standard base and openness for innovation
  - Some requirements scoped to hardware capabilities (e.g. form factors)
- Updating security requirements is one important means of improving ecosystem

Compatibility/Vendor/Security/... Test Suite
(Enforcement)

- Tests need to be run by device manufacturer
- Guaranteed conformance to (testable parts of) CDD

  In Android 10, ca. 800 tests for SELinux policy
- Usability of Android trademark and Google apps bound to passing tests
- Complexity in test execution:
  - Automation of test cases
  - Visibility on "user" firmware builds

On-device encryption

- Android 5.0 introduced **Full Disk Encryption** (FDE)
  - entangled with user knowledge factor (PIN/password), but can potentially be disabled (then encryption key only depends on device-unique key kept in TrustZone)
  - full data partition encrypted with same key, including meta data (e.g. file names)
  - all user accounts and profiles encrypted with same key
  - most system functions inaccessible until knowledge factor entered during reboot
- Android 7.0 introduced **File Based Encryption** (FBE)
  - different keys per users/profiles
  - difference between “device encrypted” (DE, only bound to unique device key) and “credential encrypted” (CE, entangled with user knowledge factor)
  - apps that are marked to use DE data storage can function after reboot before first unlock
  - Android 9 added meta data encryption
  - **Android 10 made FBE mandatory for all new devices**
  - Android 11 introduced Resume-on-Reboot

Android 10 made FBE mandatory for all new devices

User authentication

- On most mobile devices, the “lock screen” is the primary method of authentication
- (Mostly) binary distinction: locked or unlocked
  - some nuance with notifications and other information on lock screen
  - some functions can be used on locked phones (e.g. camera or emergency call)
- Can integrate with key management (KeyMaster / StrongBox)
- But implemented by Android user space ⇒ cannot defend against root adversaries

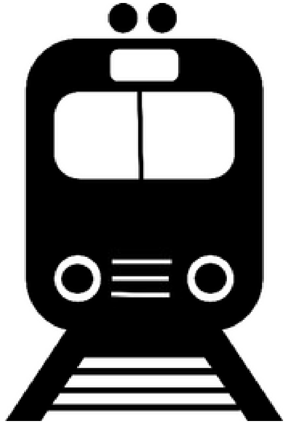
Tiered authentication model

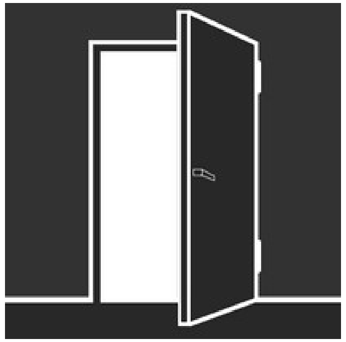
Primary Authentication
- Knowledge-factor based
- Most secure

Secondary Authentication
- Needs primary auth
- Less secure
- Somewhat constrained

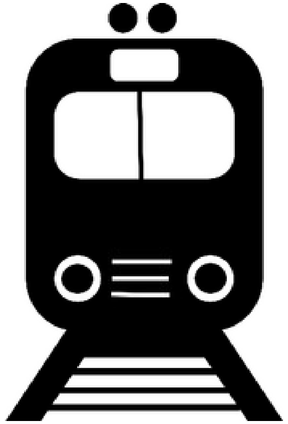
Tertiary authentication
- Needs primary auth
- Least secure
- Most constrained

Digital Authentication – Identity and Attributes

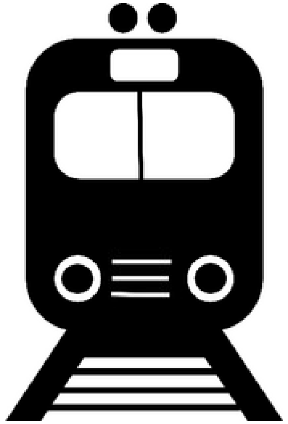




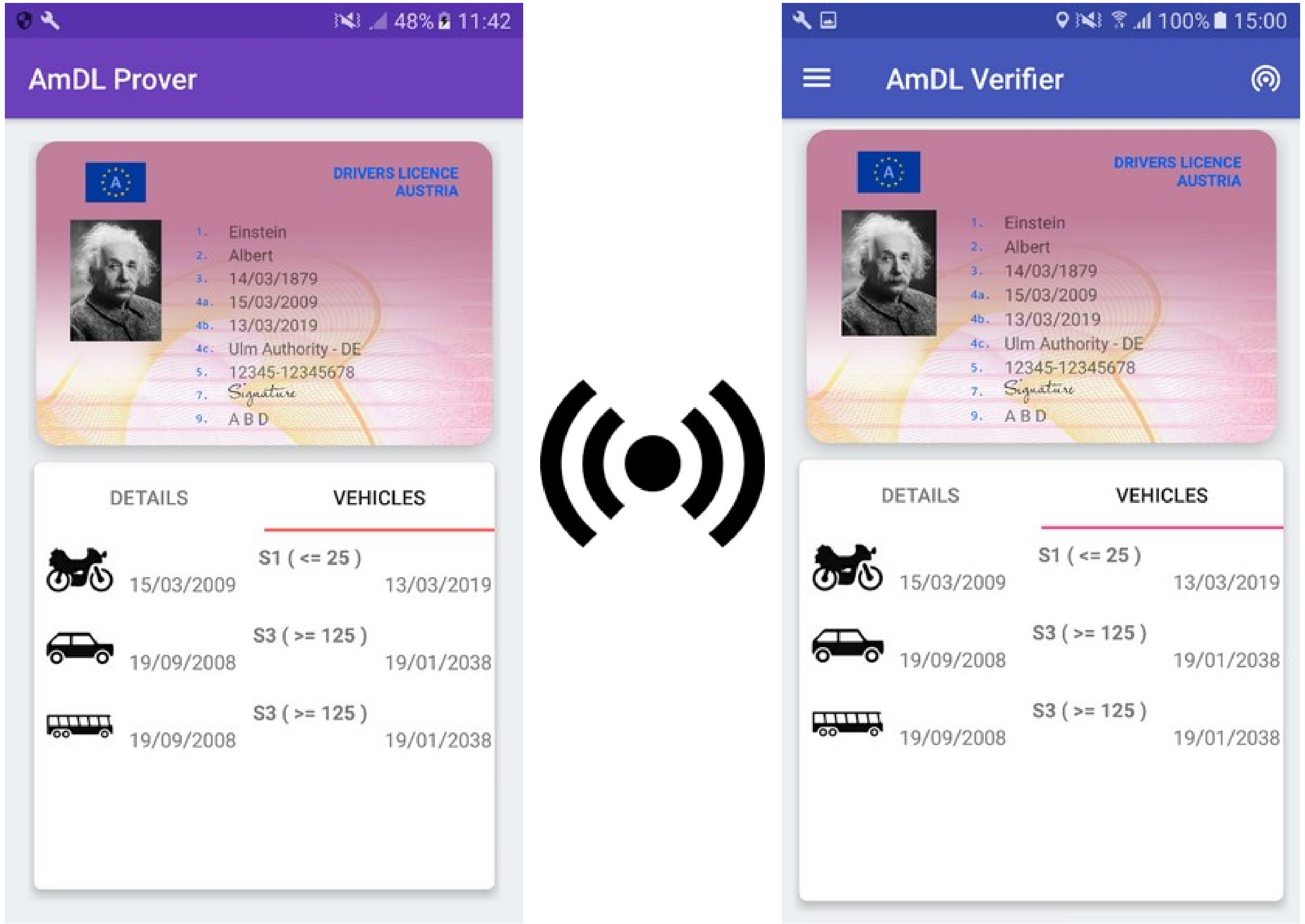
Digital Authentication – Identity and Attributes



Digital Authentication – Identity on Smartphones



Scenario 1: Traffic Check

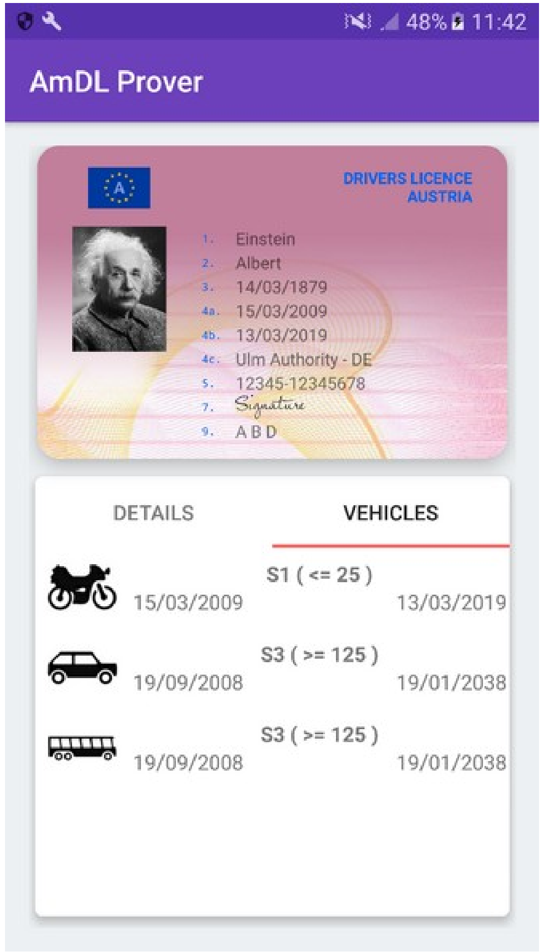


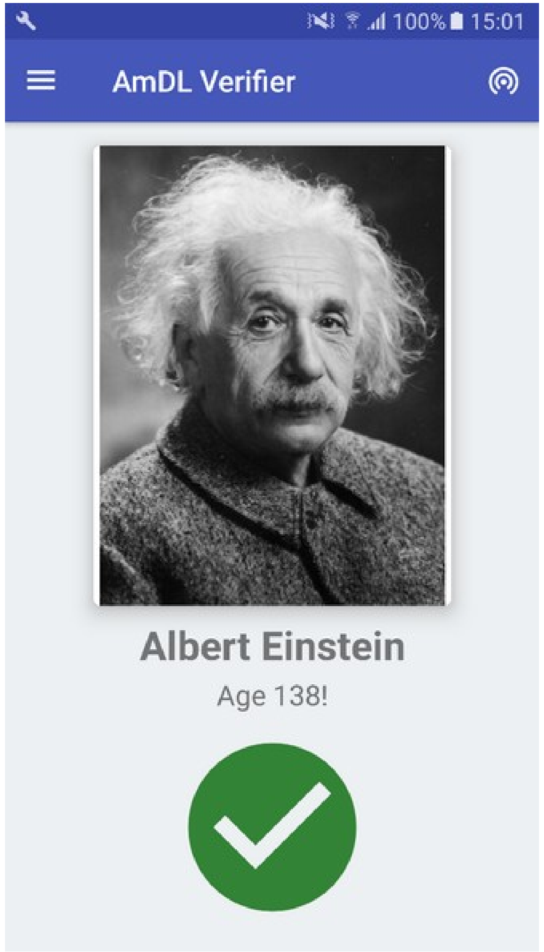
All attributes are transferred

- Name
- Date of birth
- Face picture in full resolution
- (optional) Place of residence
- (optional) Biometric features
- Vehicle classes, potential restrictions, …

Also needs to work offline!

Scenario 2: Proof of Age





Only relevant attributes

- Face picture
- Age

Scenario 3: Public Transport

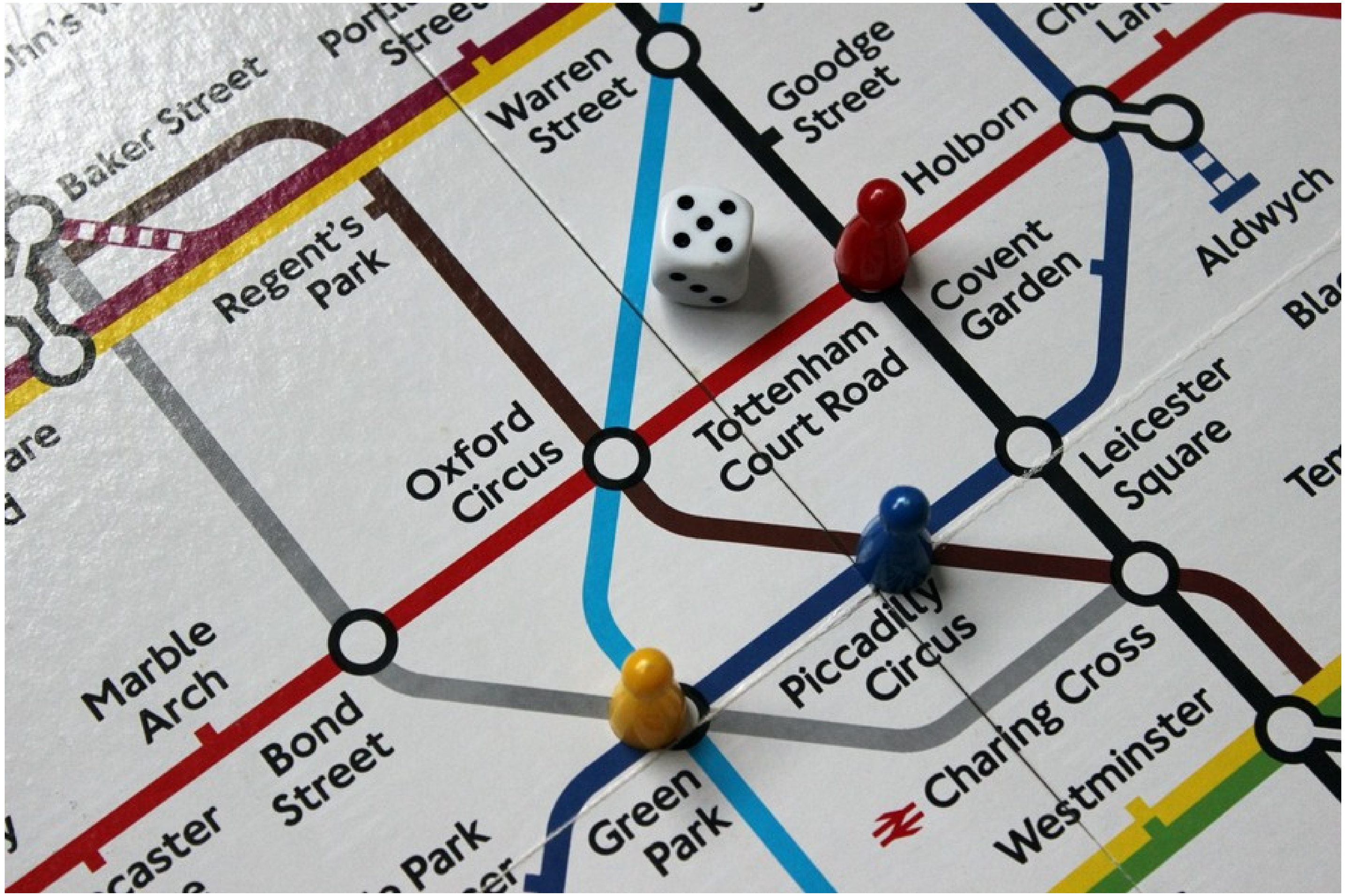


Image credit:
https://pixabay.com/photos/underground-tube-map-stations-2725336/

Location traces constitute highly sensitive data

- Place of residence / work
- Religious beliefs
- Illnesses
- Hobbies, particular preferences

Only relevant attributes

- Place of entry / exit **or**
- Possession of time based ticket

But no unique identifier!

Scenario 4: Contact Tracing

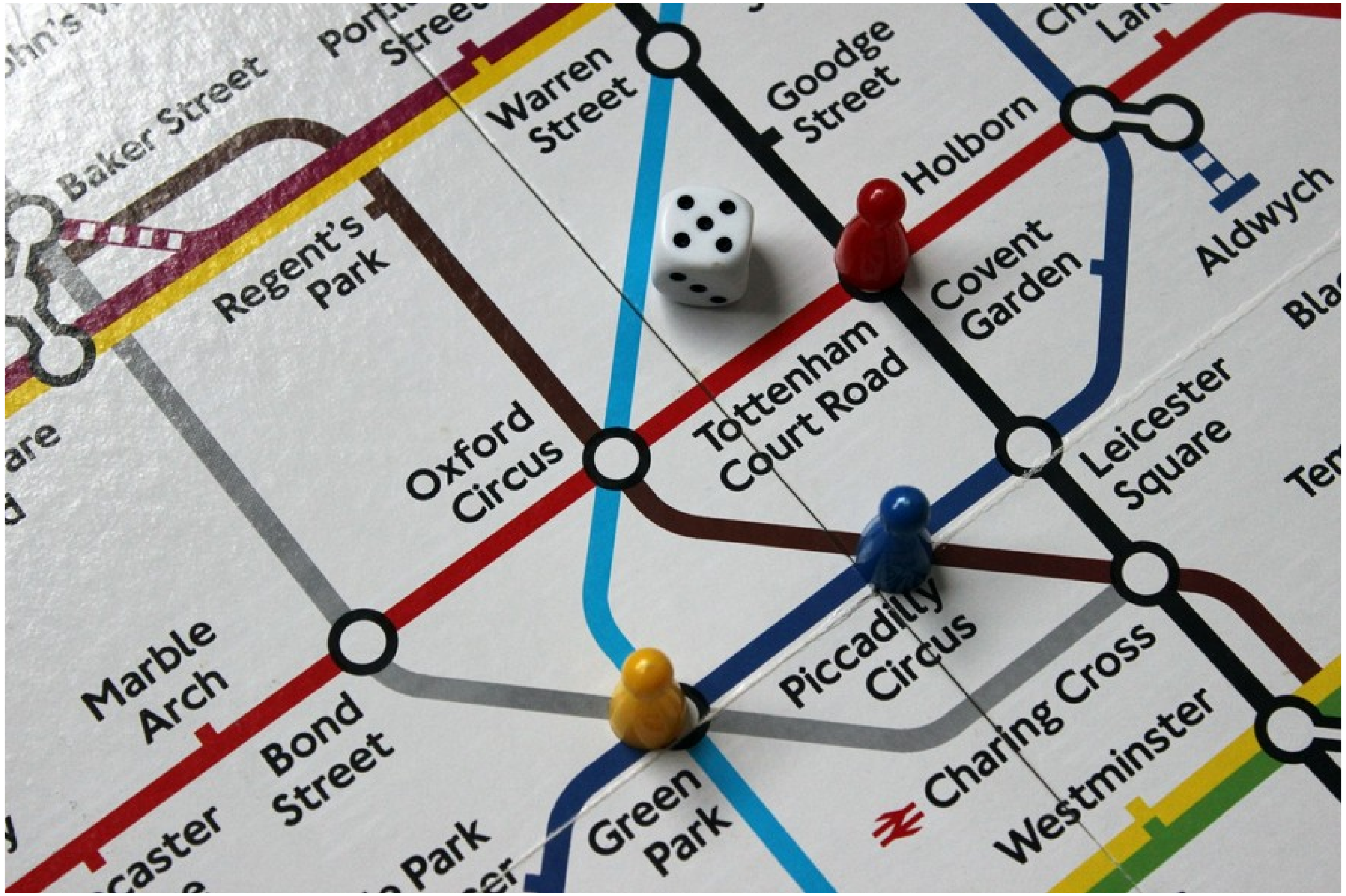


Image credit:
https://pixabay.com/photos/underground-tube-`map-stations-2725336/

Location traces constitute highly sensitive data

- Place of residence / work
- Religious beliefs
- Illnesses
- Hobbies, particular preferences

Only relevant attributes

- Contact with (pseudonym) person X for Y minutes on day Z

But no unique identifier!

JKU

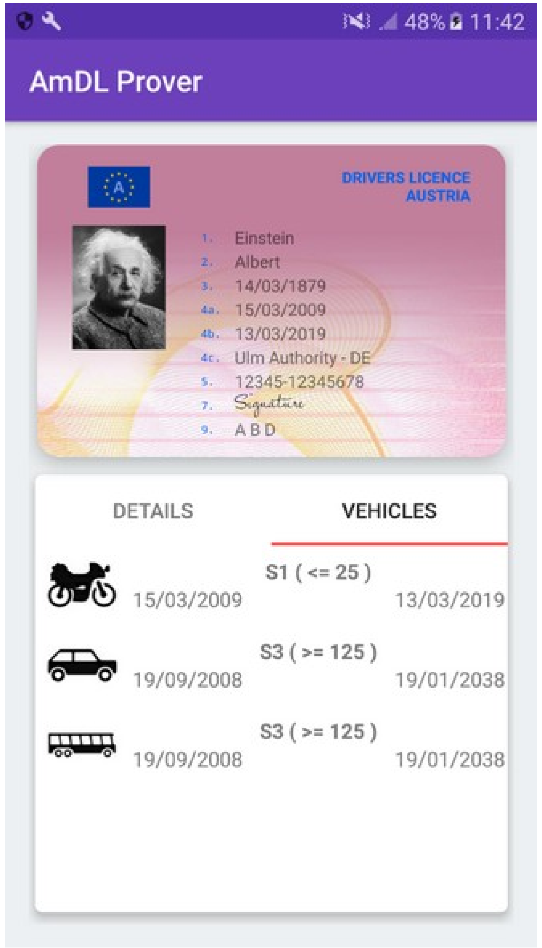
Security and Privacy for draft mDL standard (ISO 18013-5)

- **Security** properties:
  - **Anti-forgery**: Identity Credential data is signed by the Issuing Authority
  - **Anti-cloning**: Secure Hardware produces MAC during provisioning using a key derived from a private key specific to the credential and an ephemeral public key from the reader. Public key corresponding to credential private key is signed by the Issuing Authority
  - **Anti-eavesdropping**: Communications between Reader/Verifier and Secure Hardware are encrypted and authenticated
- **Privacy** properties:
  - **Data minimization**: Reader/Verifier only receives data consented to by the holder. Backend infrastructure does not receive information about use
  - **Unlinkability**: Application may provision single-use keys
  - **Auditability**: Every transaction and its data is logged and available only to the Holder (not the application performing the transaction)

Question:
Strictly require secure (certified) hardware?

J⊻U

The Android implementation



Android 11

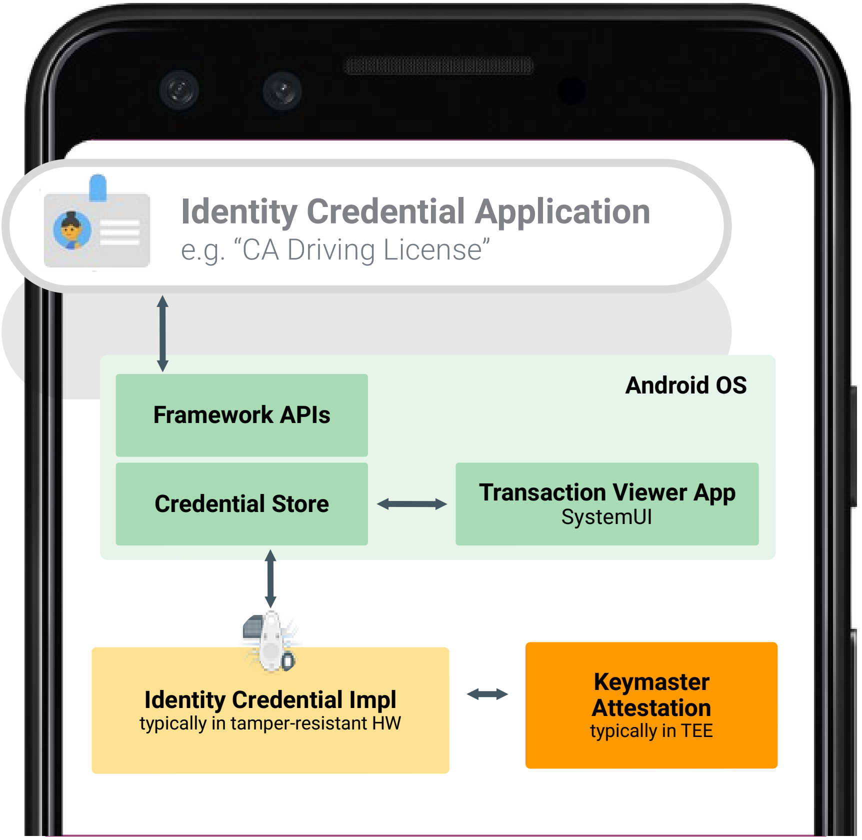


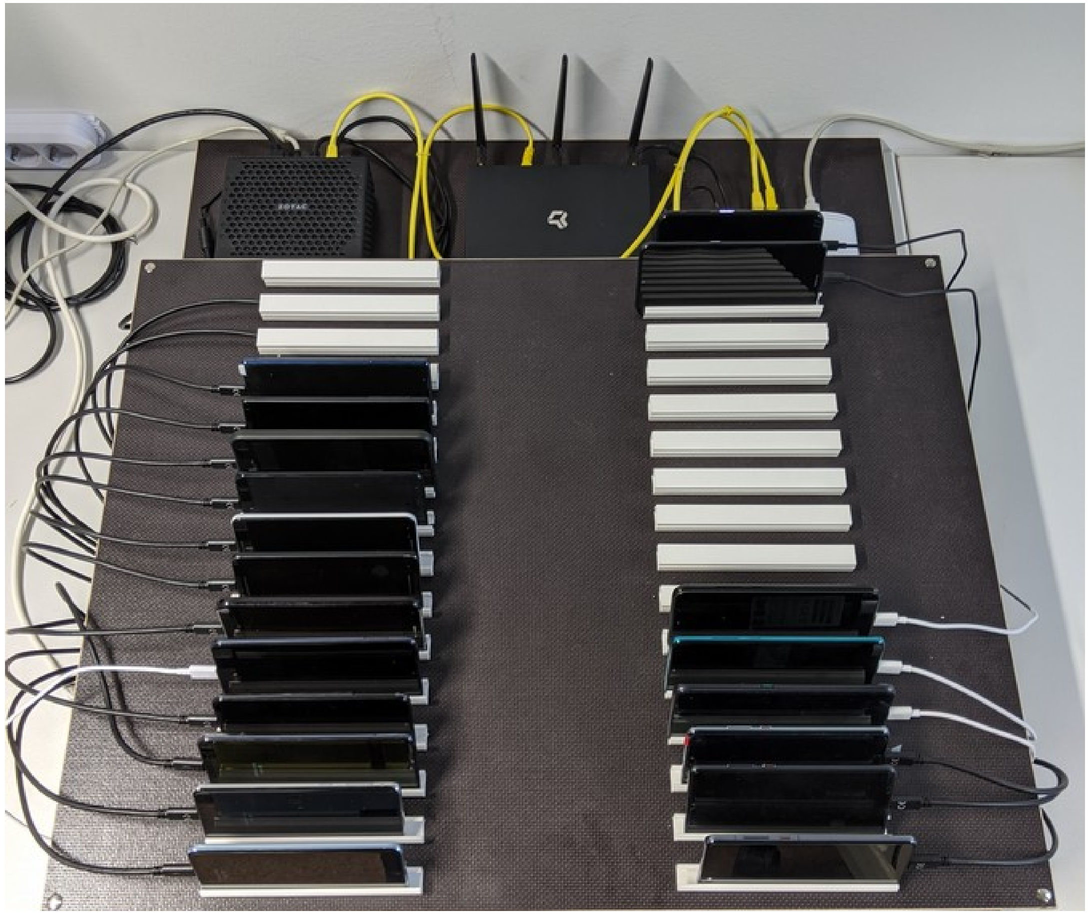
Image credit: Google

Android-Device-Security.org

- Aim: give *meaningful* data to users and organizations to make an informed decision concerning the security of a particular device
  - Provide an incentive for investing in improved security
- Collecting security attributes from devices in labs (and in the future from crowd sourcing)
  - Hardware: e.g. StrongBox support, biometric sensors, etc.
  - System/OS software: e.g. last available security patch level, multi-user support, FDE/FBE, seamless updates (A/B), etc.
  - Pre-installed apps: platform key signed, pre-granted permissions, risk level, etc.
  - Network traffic: depending on use/context, network level privacy properties (address randomization), etc.
  - Publicly documented data / OEM commitments: update support period and frequency etc.

Android-Device-Security.org: First lab at JKU Linz

- 27 different devices so far
  - Focus on European market, 9 different OEMs
  - Low-end, mid range, and flagship devices
  - Unmodified, stock system images
- Controlled through ADB
  - Reading system properties, list of apps, etc.
  - Installing test apps, collecting results
  - Daily reboot to force applying updates
- Connected through custom WiFi access point
  - One VLAN per device (selected by 802.1x)
  - Allows tracking all network traffic including layer 2 addresses (MAC randomization)



Android-Device-Security.org: Rating is hard

Image credit: https://xkcd.com/1098/

Questions?

Web: https://jku.at/ins
Email: rm@ins.jku.at
Twitter: @rene_mobile
Wire: @rm

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Straße 69
4040 Linz, Austria
jku.at