Announcements

Did you fill in the

- mid-course questionnaire?
  - https://mycourses.aalto.fi/mod/questionnaire/view.php?id=410158
- survey preferences form?
  - https://mycourses.aalto.fi/mod/questionnaire/view.php?id=406678

Deadline is today!

Lecture 3

MOBILE PLATFORM SECURITY

You will be learning:

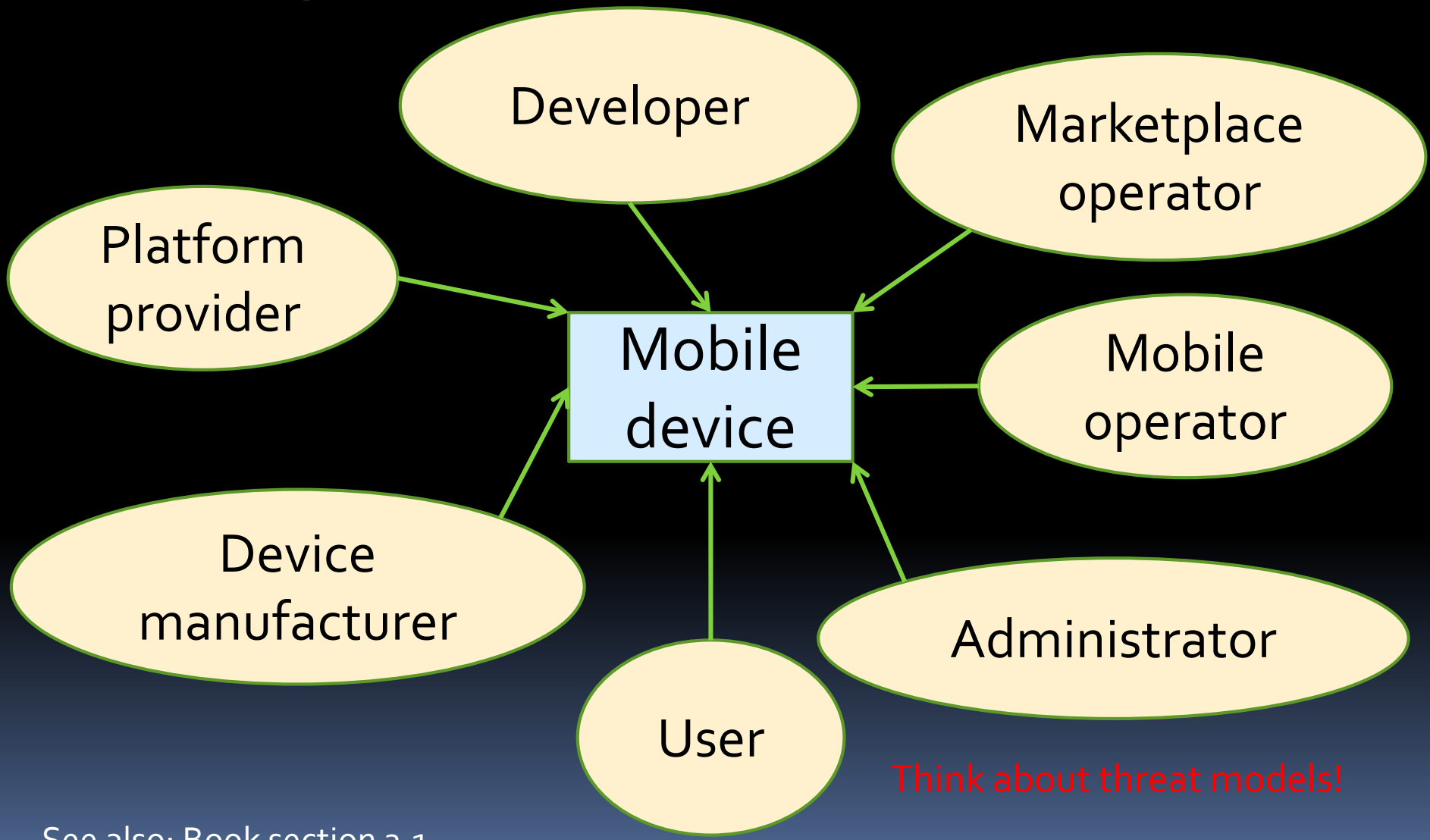
- What techniques are used in mobile software platform security?
- What techniques are used in mobile hardware platform security?
- Is there a common general architecture?

Mobile platform security

Recall classes of basic security techniques:

- Application isolation
- Permission-based access control
- Application signing
- Hardware-based security features

Ecosystem stakeholders



Think about threat models!

See also: Book section 2.1.

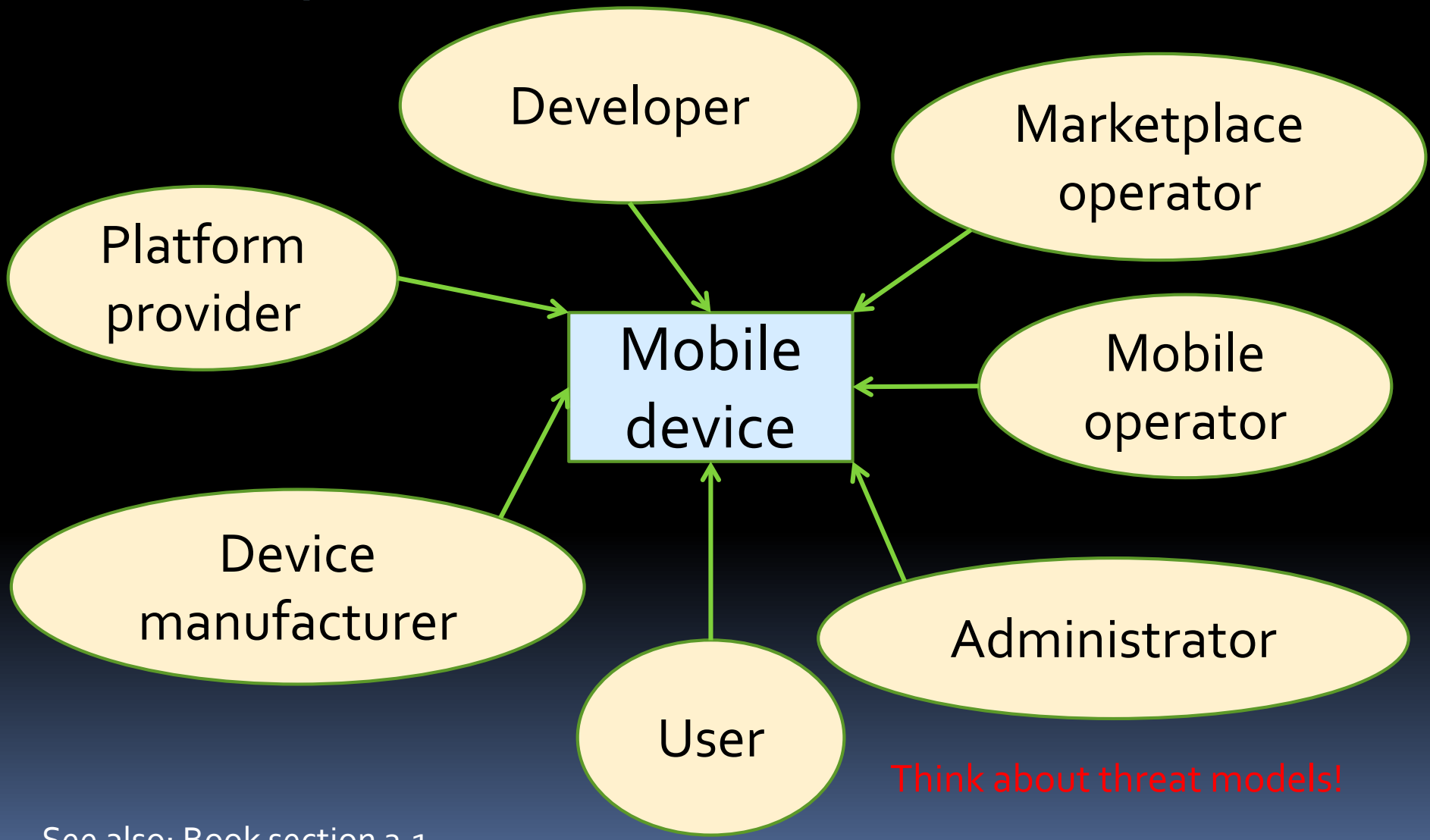
Modeling threats

Identify

- Assets and trust assumptions
- Potential adversaries
- Adversary capabilities/limitations
- Possible attack vectors

Cf. Software Security course

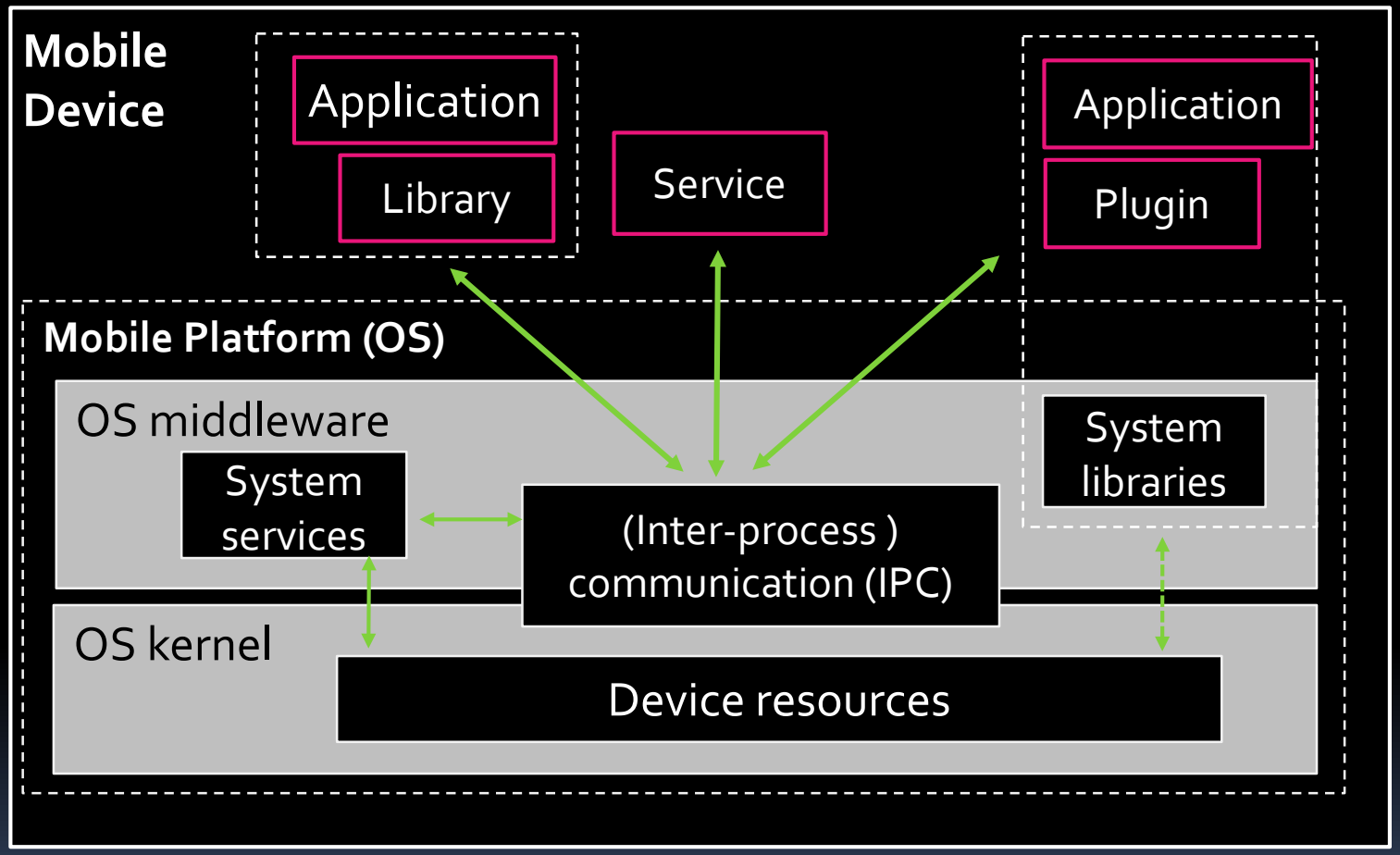
Ecosystem stakeholders



Think about threat models!

See also: Book section 2.1.

Mobile platform architecture

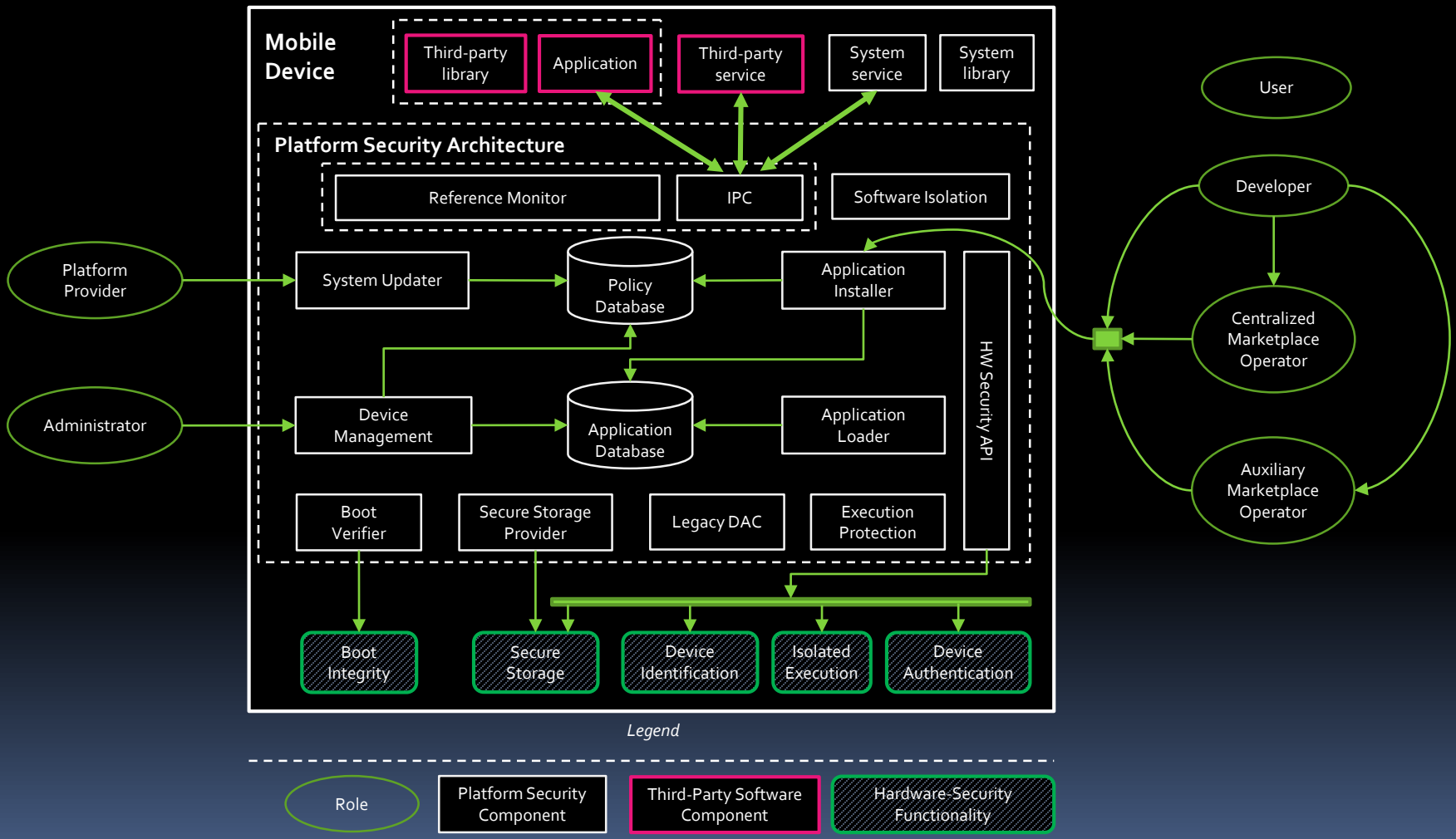


Legend

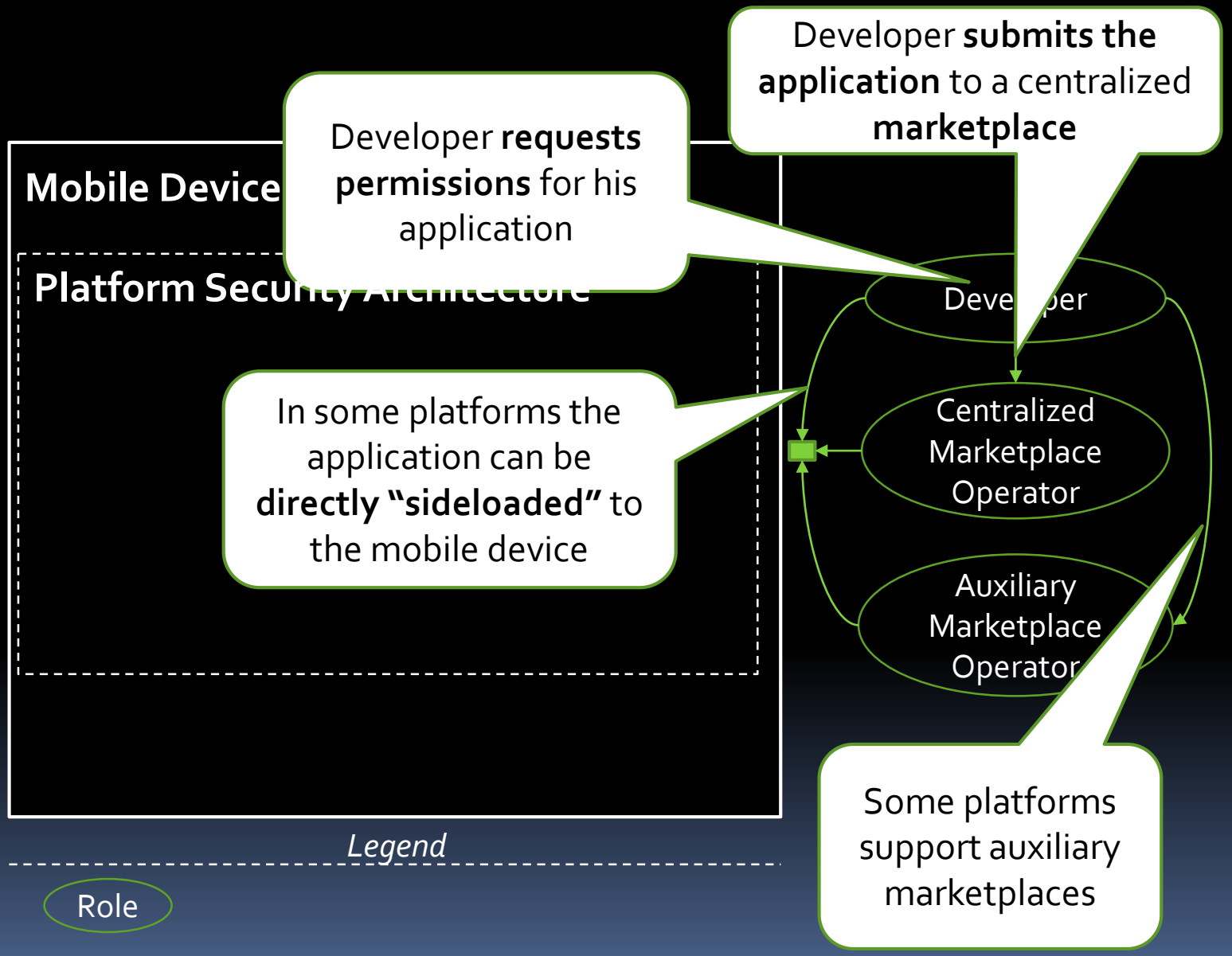
Mobile Platform Component

Third-Party Software Component

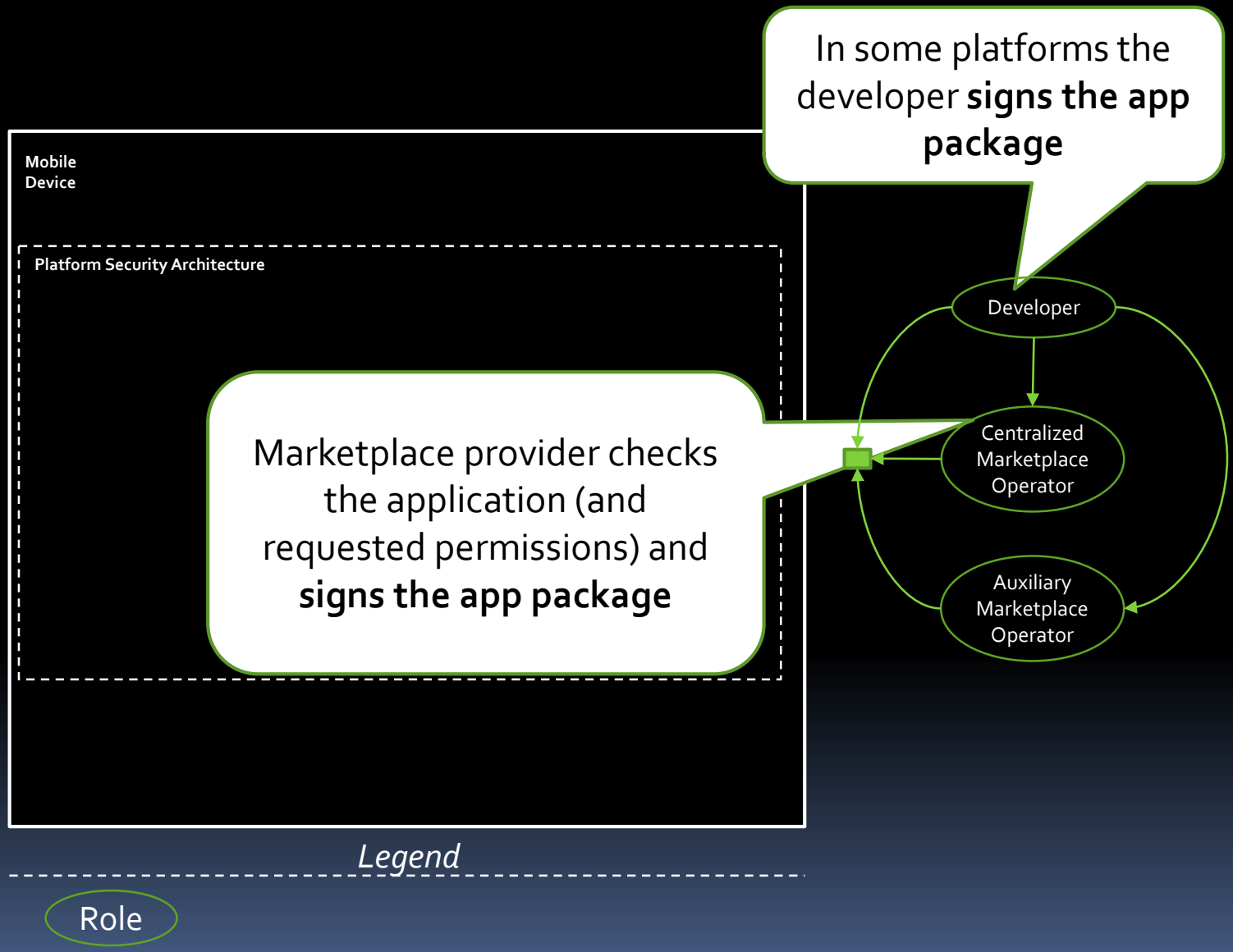
Platform security architecture



Step 1a: Developer publishes an application



Step 1b: Marketplace signs the application

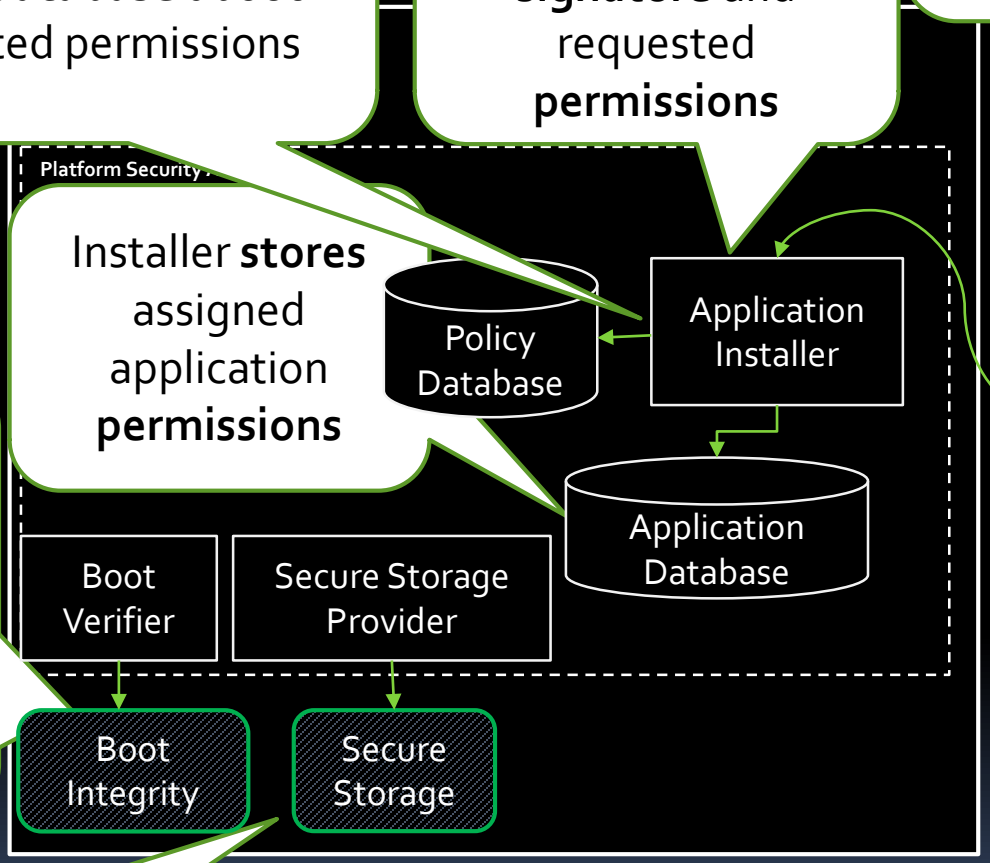


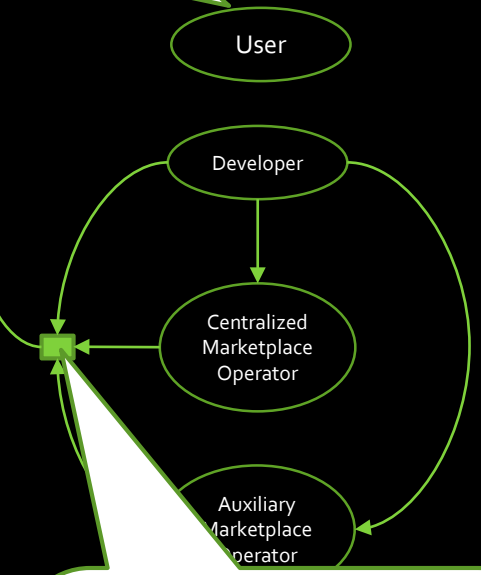
Step 2: Application installation

Installer **consults local policy database** about requested permissions

Installer **checks application signature** and requested **permissions**

Installer may **prompt** user to accept some of the requested permissions





Installer **stores** assigned application **permissions**

Application installer component needs **integrity protection**

Mobile device receives an application installation package from a marketplace (or developer)

Permission and policy databases need **integrity protection**

Legend

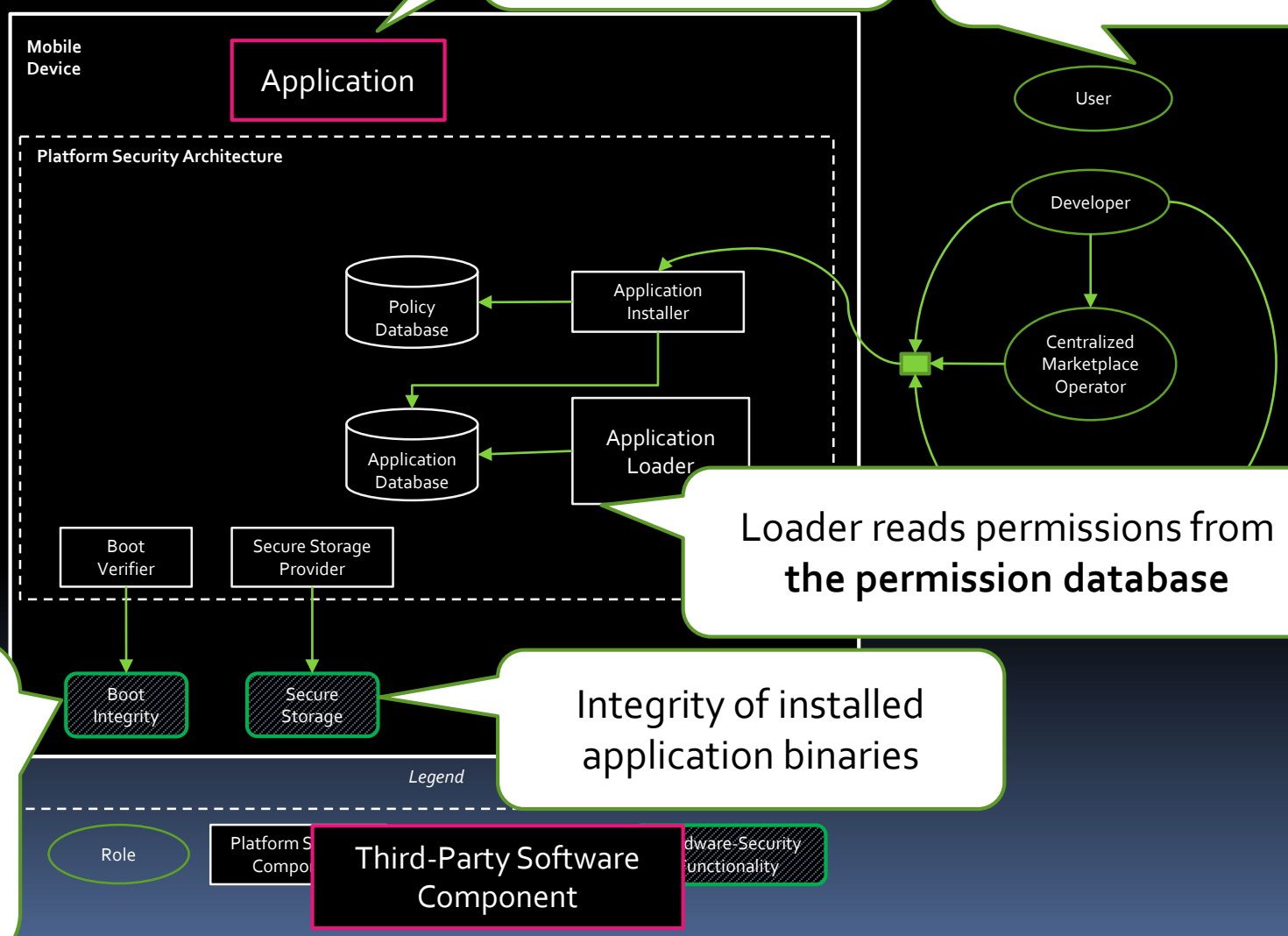
Platform Security Component

Hardware-Security Functionality

Step 3a: Application loading

Loader **attaches permissions** to the started process

Loader may **prompt** user to accept some of the requested permissions



Loader reads permissions from **the permission database**

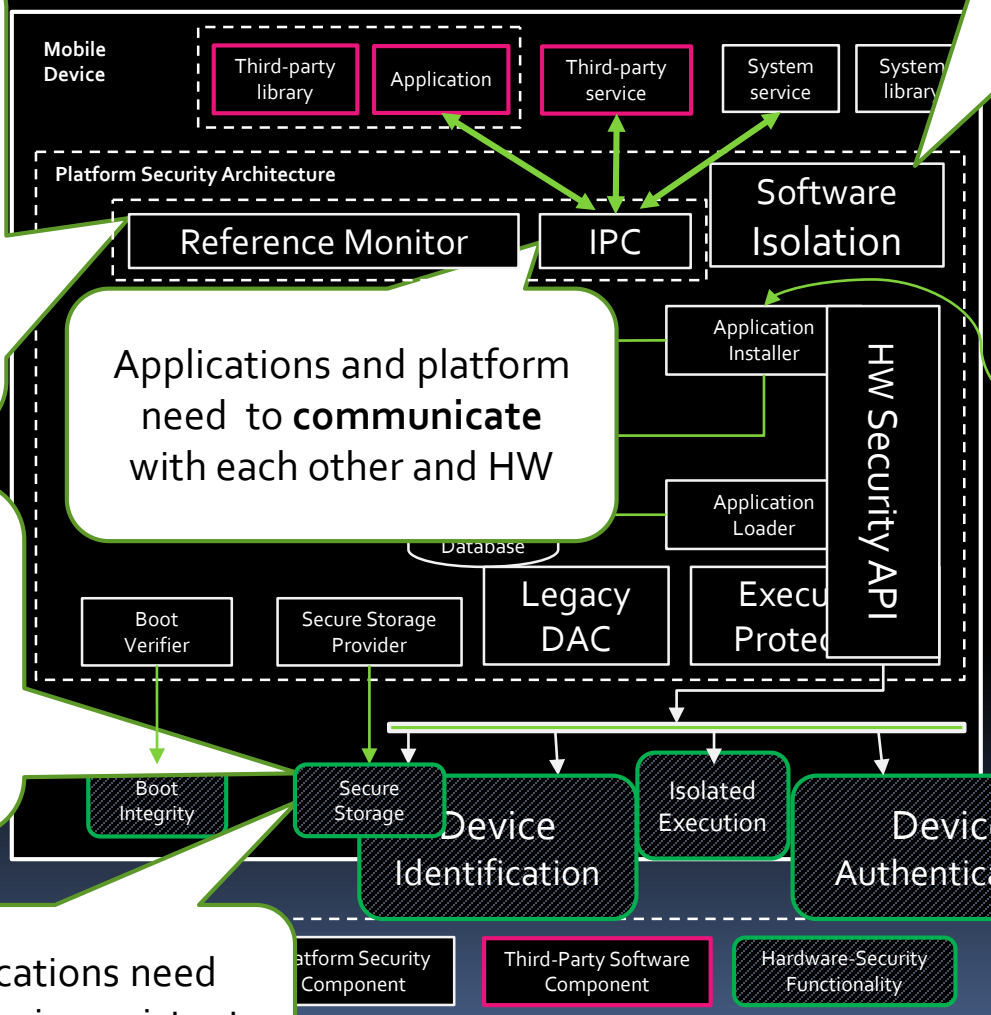
Integrity of installed application binaries

Loader component needs **integrity** protection

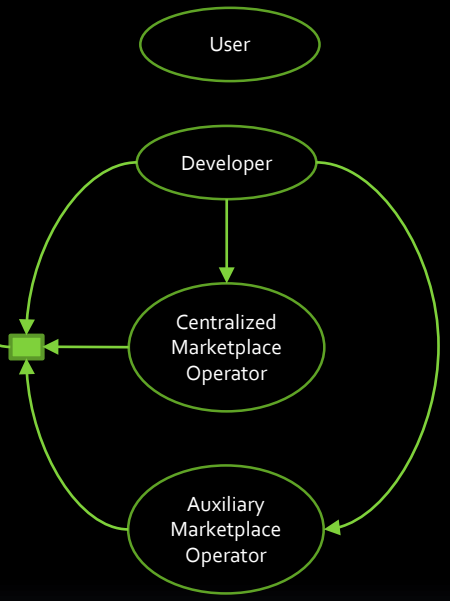
Step 3b: Application execution

OS/HW isolate applications from one another at runtime

Reference monitor checks permissions to control **access to system resources**



Applications and platform need to **communicate** with each other and HW

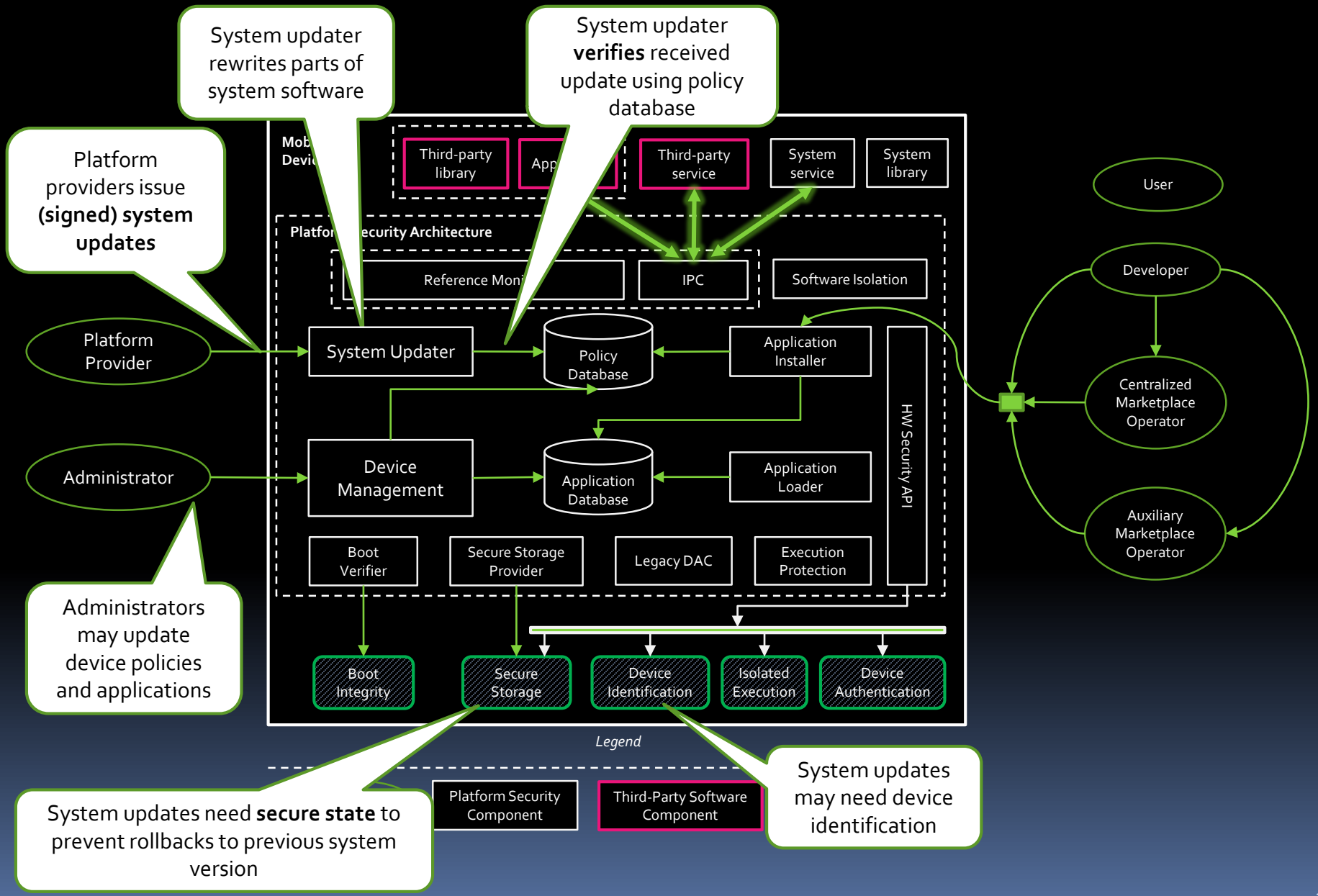


Some applications need **secure state** (e.g., DRM)

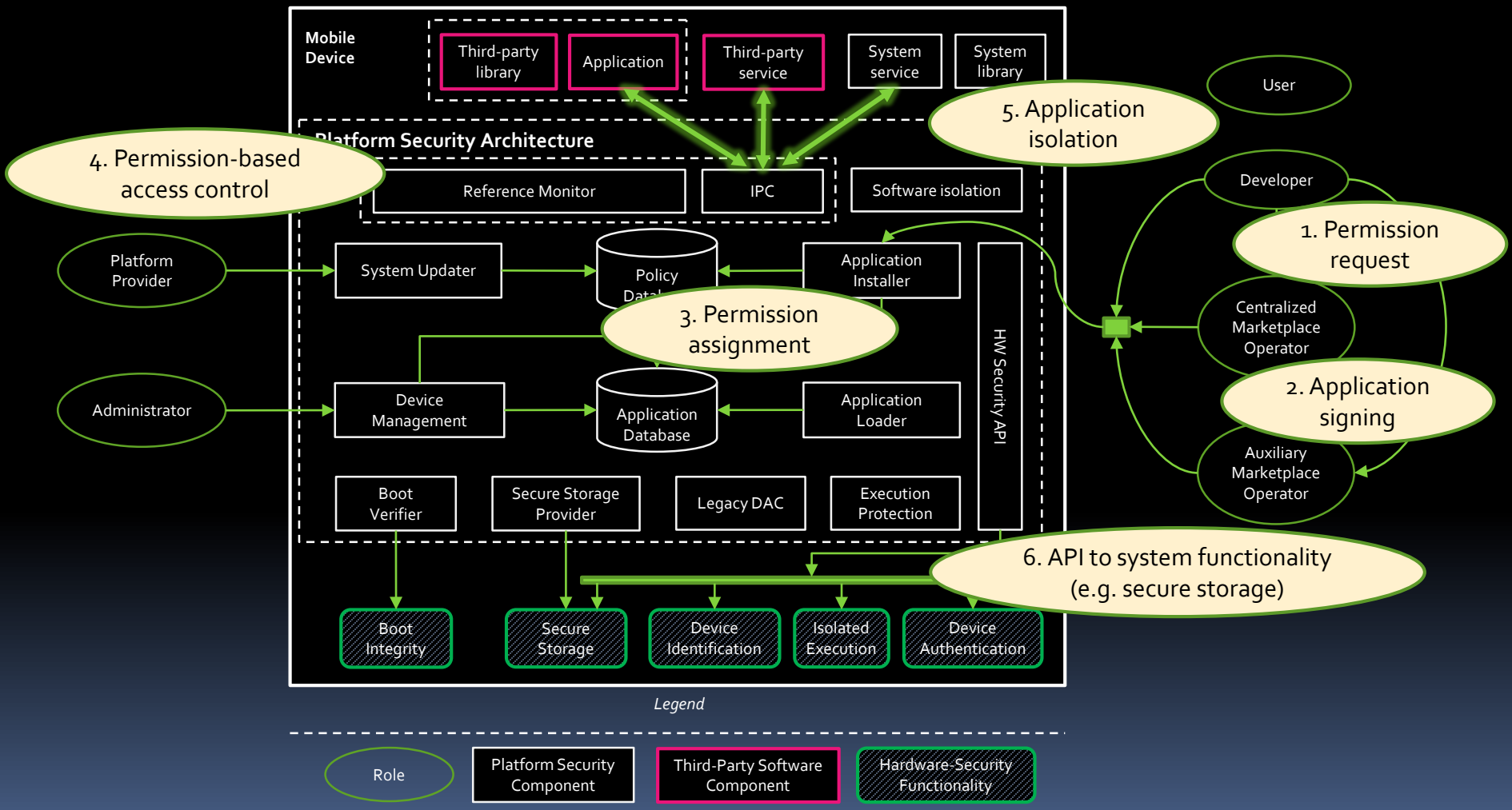
Some applications need **secrecy** for their persistent storage

Some applications need **device identification and authentication** (e.g., provisioning)

Step 4: System updates

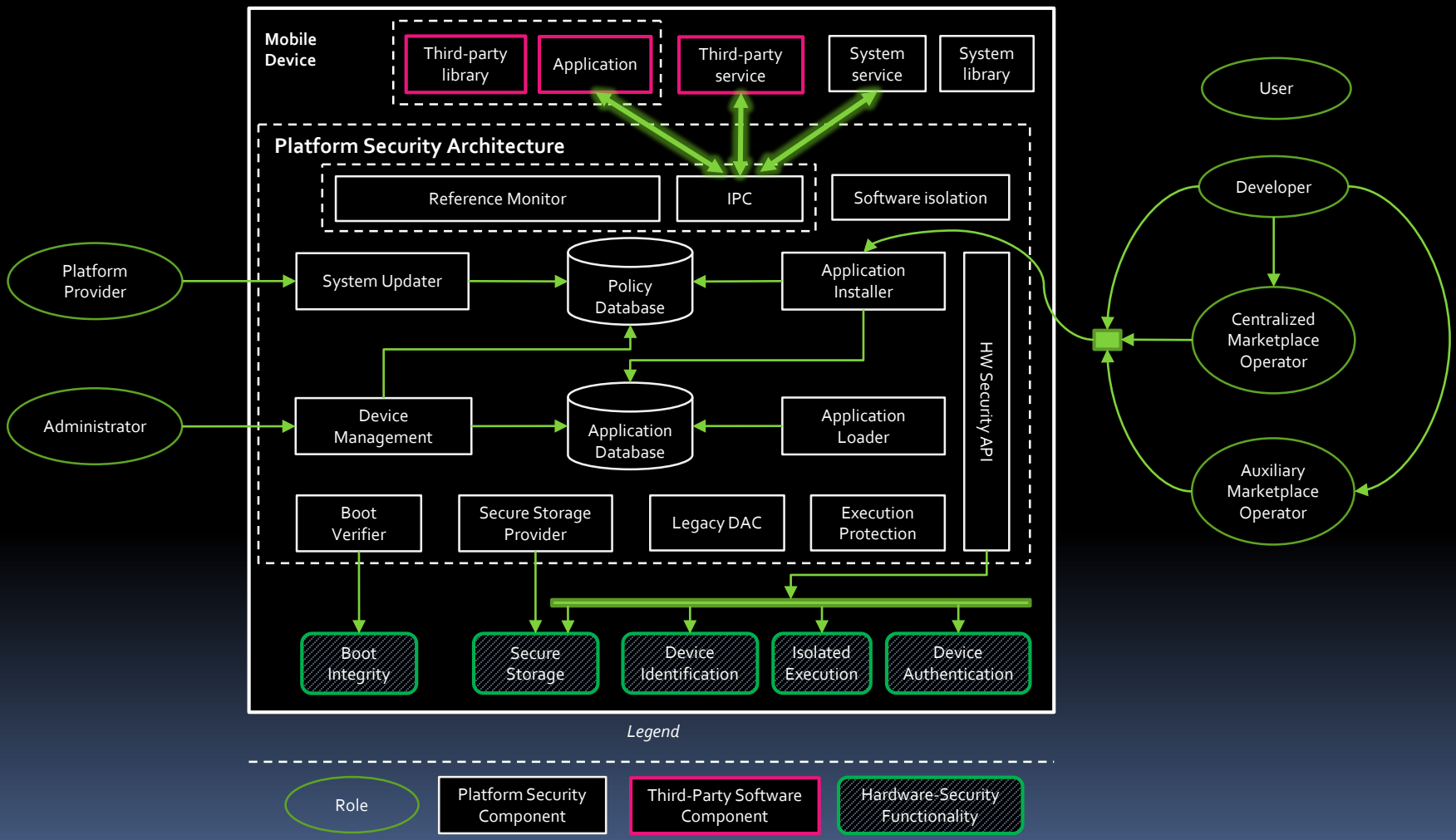


Recap: main techniques



Skip to other OSs

Platform security architecture



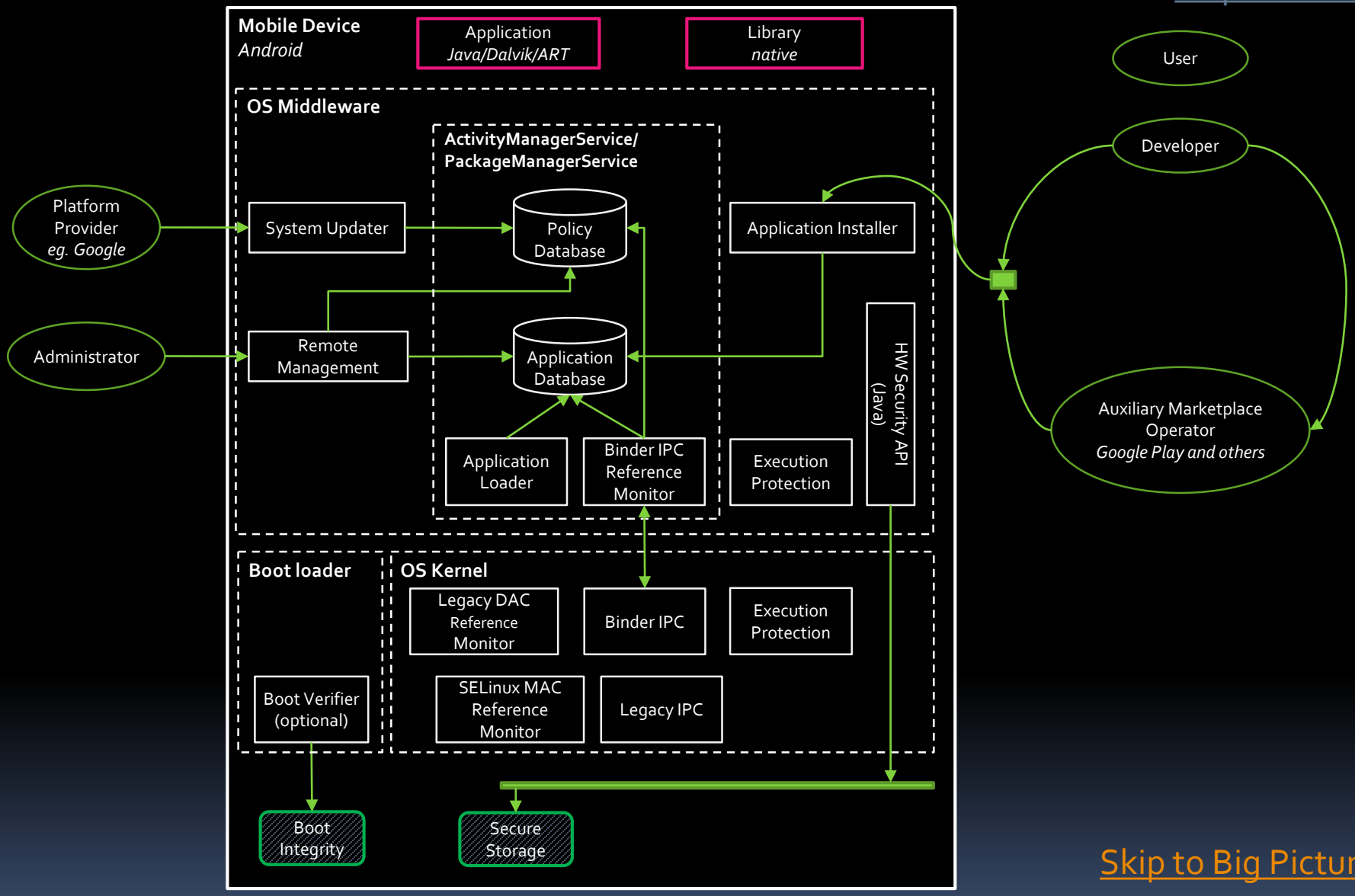
Mobile platforms revisited

- Android ~2007
- Java ME ~2001
  - "feature phones": 3 billion devices!
  - Not in smartphone platforms
- Symbian ~2001
  - First "smartphone" OS

Mobile platforms revisited

- iOS ~2007
  - iP* devices; BSD-based
- MeeGo ~2010
  - Linux-based
  - MSSF (security architecture)
- Windows Phone ~2010
- …

Skip to Model



Skip to Big Picture

Legend

Role

Platform Security Component

Third-Party Software Component

Hardware-Security Functionality

Symbian

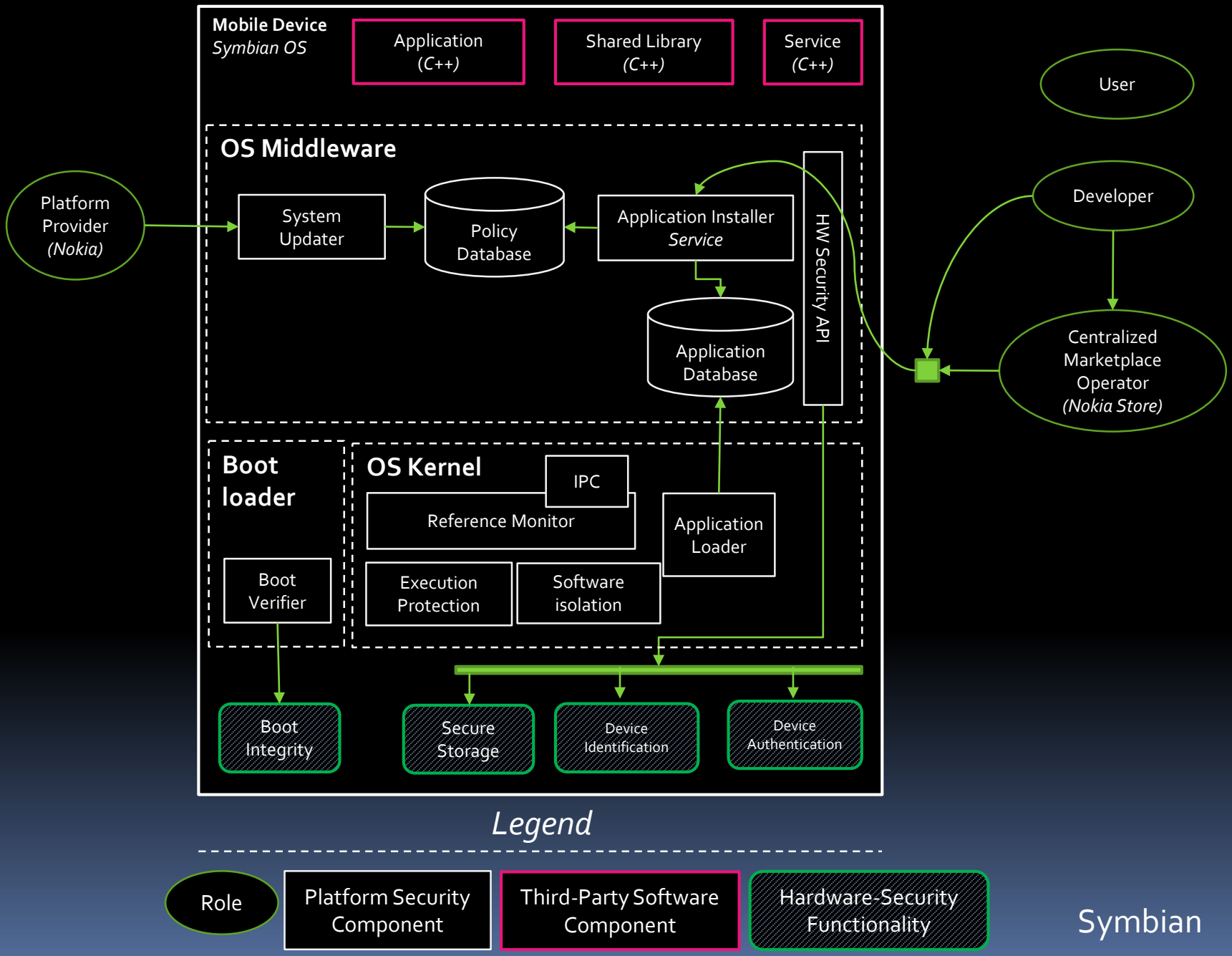
- First widely deployed smartphone OS
  - EPOC OS for Psion devices (1990s)
- Microkernel architecture:
  - OS components as user space services
  - Accessed via inter-process communication (IPC)

Symbian Platform Security

- Introduced in ~2004
- Apps distributed via Nokia Store
  - Sideloading supported
- Permissions are called ‘capabilities’, fixed set (21)
  - 4 Groups: User, System, Restricted, Manufacturer

Symbian Platform Security

- Applications identified by:
  - UID from protected range, based on trusted code signature
  - Or UID picked by developer from unprotected range
  - Optionally, vendor ID (VID), based on trusted code signature


Mobile Device
Symbian OS
Application (C++)
Shared Library (C++)
Service (C++)
OS Middleware
Platform Provider (Nokia)
System Updater
Policy Database
Application Installer Service
HW Security API
Application Database
User
Developer
Centralized Marketplace Operator (Nokia Store)
Boot loader
Boot Verifier
OS Kernel
IPC
Reference Monitor
Application Loader
Execution Protection
Software isolation
Boot Integrity
Secure Storage
Device Identification
Device Authentication
Legend
Role
Platform Security Component
Third-Party Software Component
Hardware-Security Functionality

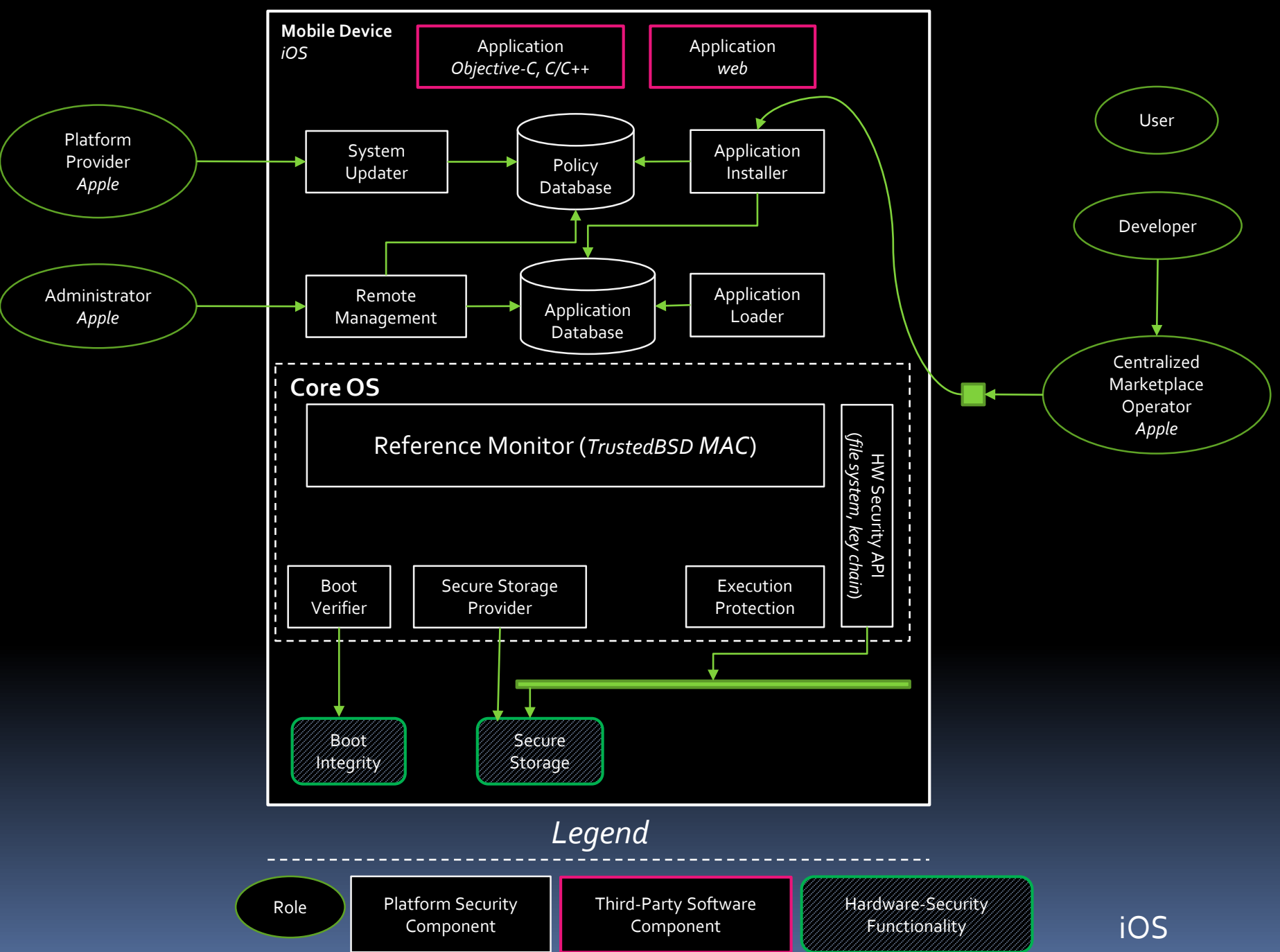
Symbian

Apple iOS

- Native application development in Objective C
  - Web applications on Webkit
- Based on Darwin + TrustedBSD kernel extension
  - TrustedBSD implements Mandatory Access Control
  - Darwin also used in Mac OS X

iOS Platform Security

- Apps identified by unique "app IDs"
  - Cf. Android package names
- Apps distributed via iTunes App Store
- One centralized signature authority
  - Apple software vs. third party software
- Runtime protection
  - All 3rd party s/w sandboxed with same profile
  - Permissions: "entitlements" (post iOS 6)
  - Contextual permission prompts: e.g. location


Mobile Device
iOS
Application
Objective-C, C/C++
Application
web
Platform Provider
Apple
System Updater
Policy Database
Application Installer
User
Developer
Administrator
Apple
Remote Management
Application Database
Application Loader
Centralized Marketplace Operator
Apple
Core OS
Reference Monitor (TrustedBSD MAC)
HW Security API
(file system, key chain)
Boot Verifier
Secure Storage Provider
Execution Protection
Boot Integrity
Secure Storage
Legend
Role
Platform Security Component
Third-Party Software Component
Hardware-Security Functionality

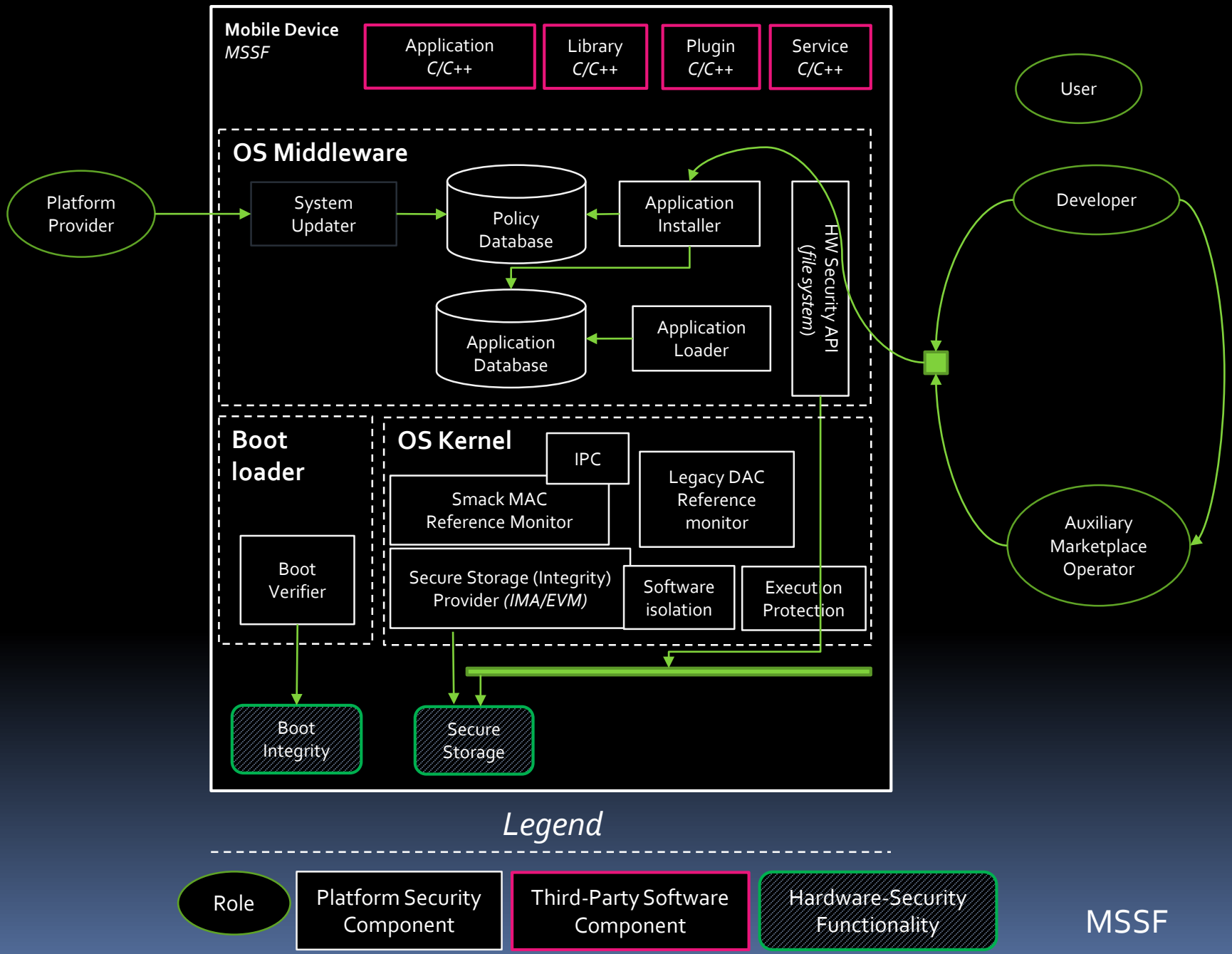
iOS

MeeGo

- Linux-based open source OS
  - Intel, Nokia, Linux Foundation
  - Evolved from Maemo and Moblin
- Application development in Qt/C++
- Partially buried, but lives on
  - Linux Foundation shifted to HTML5-based Tizen
  - MeeGo -> Mer -> Jolla's Sailfish OS

MeeGo Platform Security

- Mobile Simplified Security Framework (MSSF)
  - Permissions: “resource tokens”
  - Enforced via “Smack”
  - Apps identified by signatures from “software sources”
  - Policy specifies privileges grantable by software sources


Mobile Device
MSSF
Application
C/C++
Library
C/C++
Plugin
C/C++
Service
C/C++
OS Middleware
Platform Provider
System Updater
Policy Database
Application Installer
HW Security API
(file system)
Application Database
Application Loader
Boot loader
OS Kernel
IPC
Smack MAC Reference Monitor
Legacy DAC Reference monitor
Boot Verifier
Secure Storage (Integrity) Provider (IMA/EVM)
Software isolation
Execution Protection
Boot Integrity
Secure Storage
User
Developer
Auxiliary Marketplace Operator
Legend
Role
Platform Security Component
Third-Party Software Component
Hardware-Security Functionality
MSSF

Skip to App Installation

Skip to Big Picture

Model for platform security

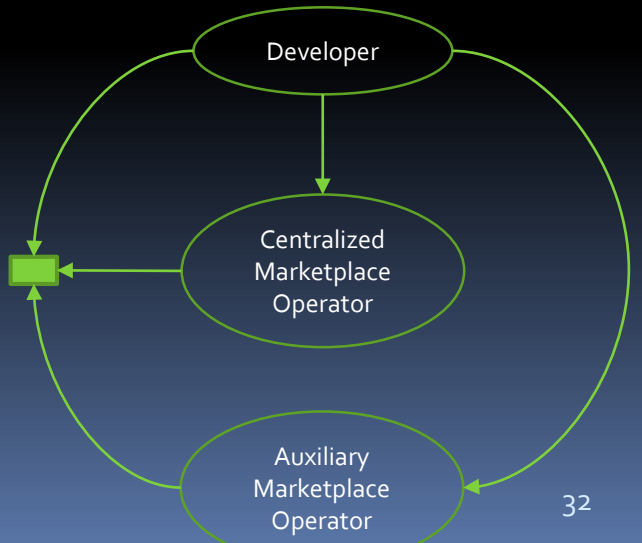
Four processes to protect:

1. Software deployment
2. Application installation
3. Runtime operation
4. Platform management

1. Software deployment

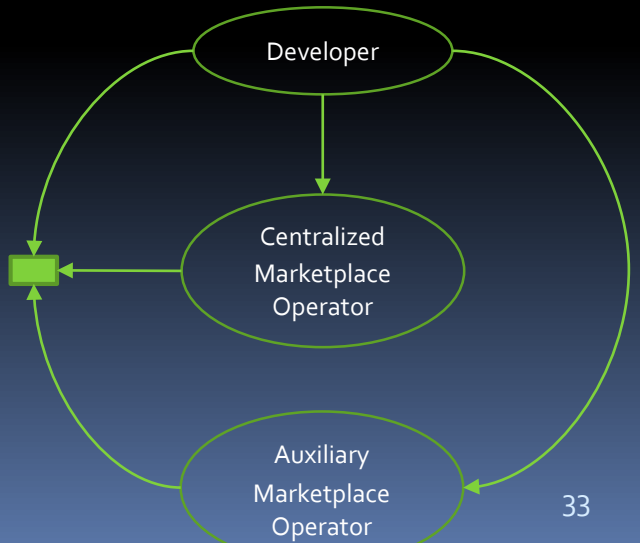
Developing and publishing

- Design choices:
  - Distribution : centralized vs. decentralized
  - App signing: certified vs. self-signed



1. Software deployment

- Design choices:
  - App identification: global vs. local
  - …



More design choices discussed in book chapter 4.

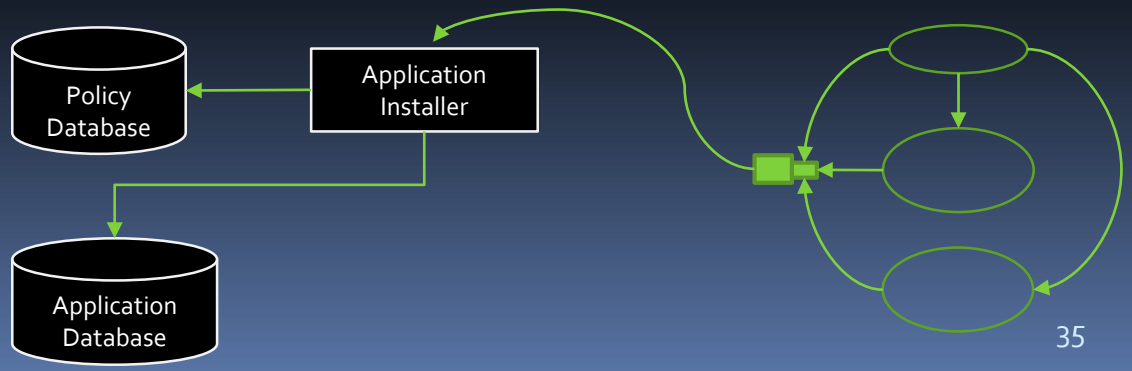
Software deployment

	Android	iOS	MSSF	Symbian
Distribution model	Multiple marketplaces, sideloading	Centralized marketplace	Multiple marketplaces, sideloading	Centralized marketplace, limited sideloading
Application signing	Developer signature	Centralized signature	Marketplace and developer signature	Centralized or developer signing: affects set of permissions
Application identifier	Package ID, local Linux UID for permissions	Application ID	3-part ID: Marketplace - package - application	Application ID, vendor ID

2. App installation

Acquiring/installing a new app

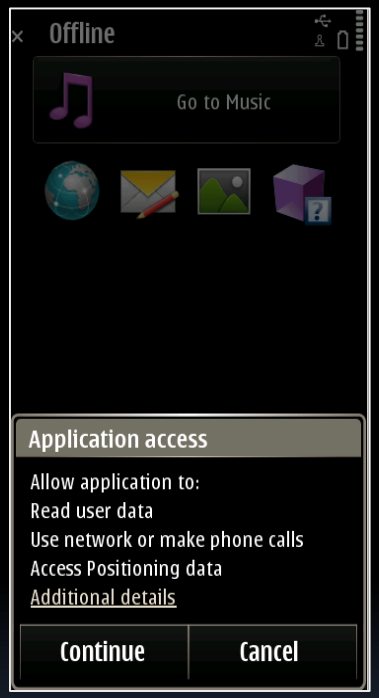
- Design choices:
  - Permission assignment: user vs. authority?
  - Permission granularity?
  - Application updates: same origin vs. centrally authorized?



Skip to Big Picture

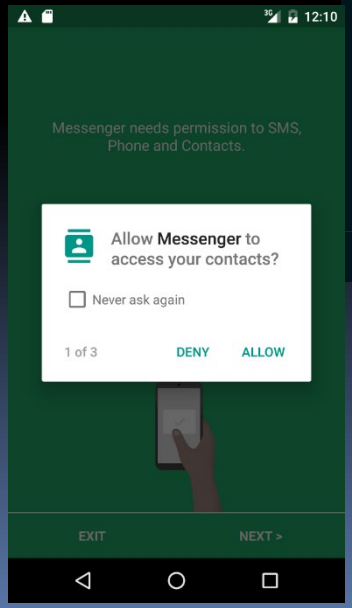
App permissions

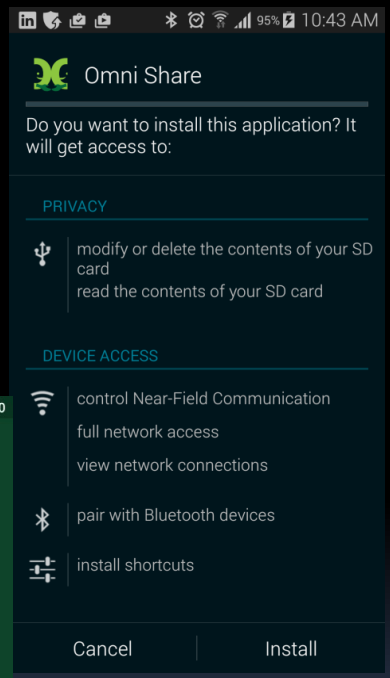
- Ask user?
  - Install or use time?
- Automatic granting?
- Revocation?
- What about libraries?



Symbian

iOS





Android

Skip to Big Picture

App permissions

	Android	iOS	MSSF	Symbian
Granularity	Fine-grained	Pre-defined profiles (iOS6 entitlements)	Fine-grained	Coarse-grained
Assignment	Ask user or app signature	Fixed profile for all apps	Marketplace-specific rights profiles	Ask user or centralized signature
Ask user: presentation	by group (11), install time & runtime (6.0)	by name, runtime	never ask	by name (21), install time

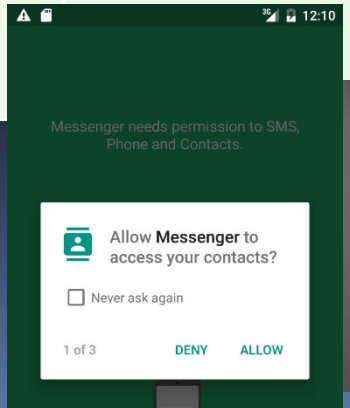
Both Android (> 6.0) and iOS allow revocation of granted permissions

Skip to Big Picture

App permissions

- Runtime permission changes

	Android	iOS	MSSF	Symbian
Changes in process permissions at runtime	Pre 6.0: Constant (except URI permissions) > 6.0: User can change rights	User can change rights	Rights can increase by plugin loading, **decrease by request**	Constant (library loading can fail)
Permissions of libraries	App's permissions	App's permissions	Union of app and library permissions	App's permissions (library perms must be a superset)



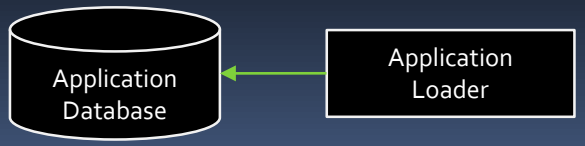
App updates

- Who can update an app?
  - Same-origin: same dev. key
  - Trusted marketplace(s)
  - Allow anyone

	Android	iOS	MSSF	Symbian
Updates allowed if…	Same-origin: must match old version's developer key	Centrally signed	Marketplace's trust level high enough	Protected? Same-origin; Unprotected? Anyone

3. Runtime operation

- Design choices:
  - Permission enforcement: where?
  - App data protection: how to secure storage?



Hardware support: Later in Lecture 3, Lecture 4

Runtime operation

- Access control enforcement: where is "reference monitor"?

	Android	iOS	MSSF	Symbian
Where is access control enforced?	UID/GID-based in kernel + IPC access control in Binder + application code	Centralized	D-Bus framework + socket IPC in kernel + application code	Reference monitor for IPC calls + application code

Protecting data & code

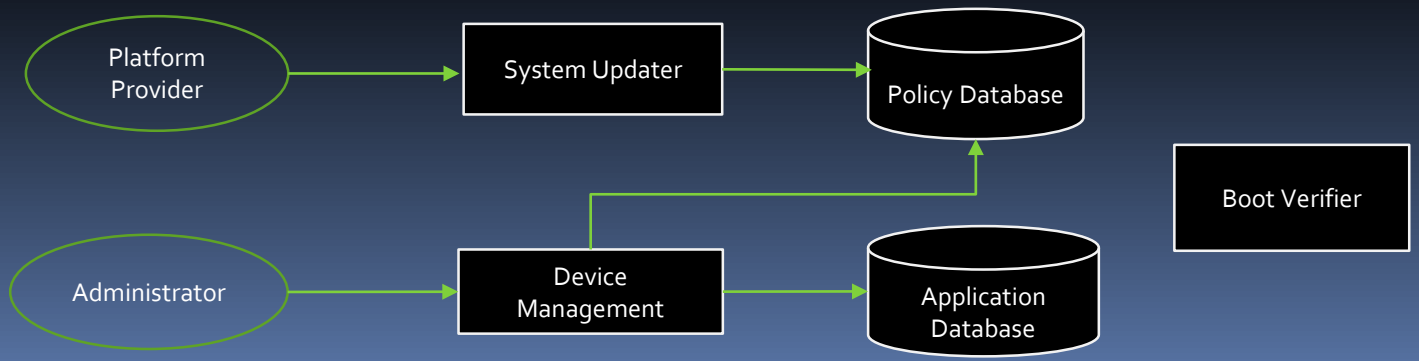
- Applications: isolation for data access
- Platform: executables (see later)

	Android	iOS	MSSF	Symbian
Application data integrity	Own directory and Linux access control	Access to own directory only	Permission-based policies	Own directory

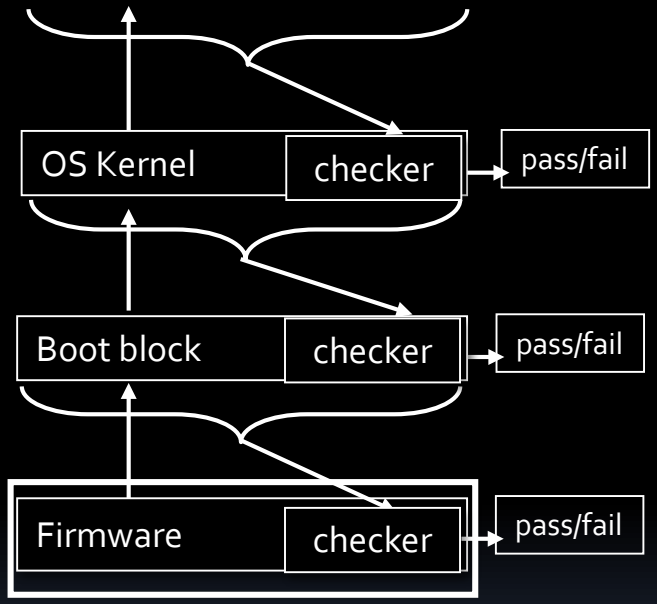
4. Platform management

Bootup, platform integrity, updates

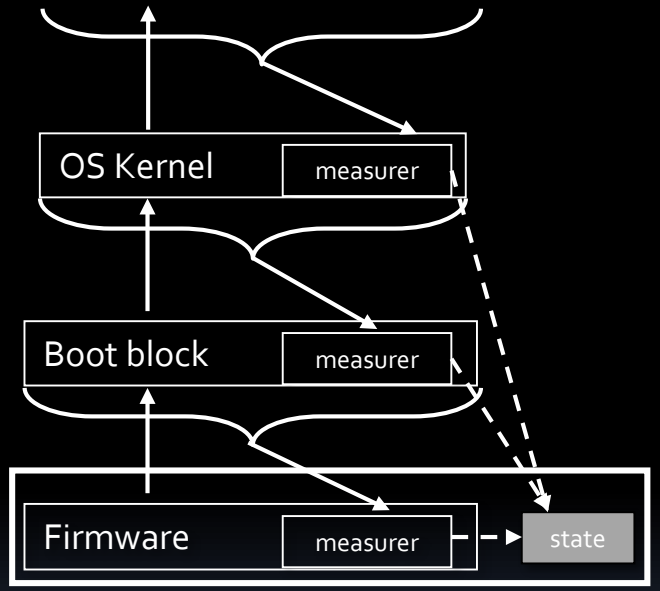
- Design choices:
  - Boot integrity: secure vs. authenticated?



Secure boot vs. authenticated boot



Secure boot



Authenticated boot

Boot integrity

- Secure boot
  - Only authorized images can be booted
- Authenticated boot
  - Access levels depend on booted image

	Android	iOS	MeeGo	Symbian
Platform boot integrity	Vendor-specific, verified boot	Secure boot	Secure boot, authenticated boot	Secure boot

Platform data integrity

	Android	iOS	MeeGo	Symbian
Platform data integrity	Linux A/C, SELinux, UID-based sandboxing	Dedicated directory, code signing enforcement	Linux A/C, Smack, IMA, EVM	Dedicated directory

- IMA: Integrity measurement architecture
- EVM: Extended validation module

(see "An Overview of The Linux Integrity Subsystem")

The big picture

Recurring common themes

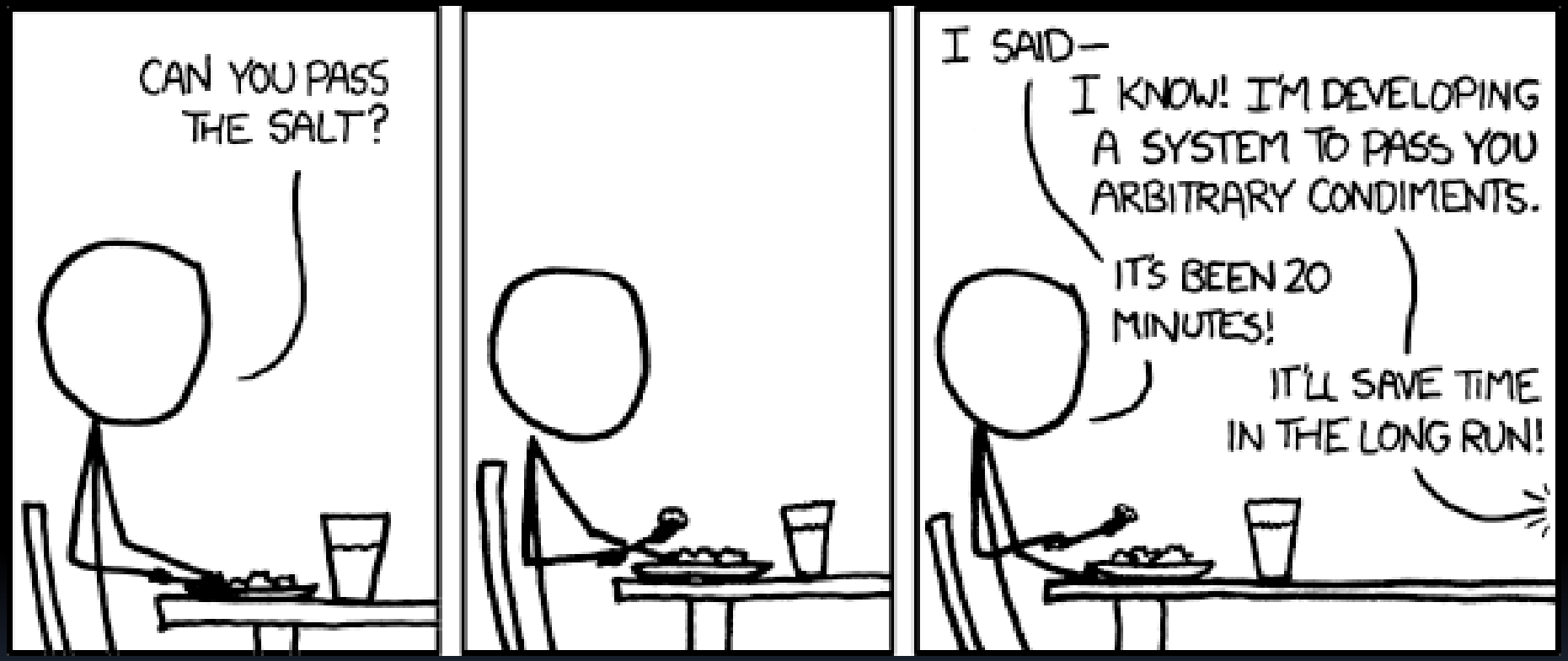
- Permission-based security architectures
  - VAX /VMS privileges (~1970's): adapted for applications
  - Code signing (mid 1990's): adapted for application installation
- Application/process isolation

The big picture

Different choices in the design space lead to different architectures

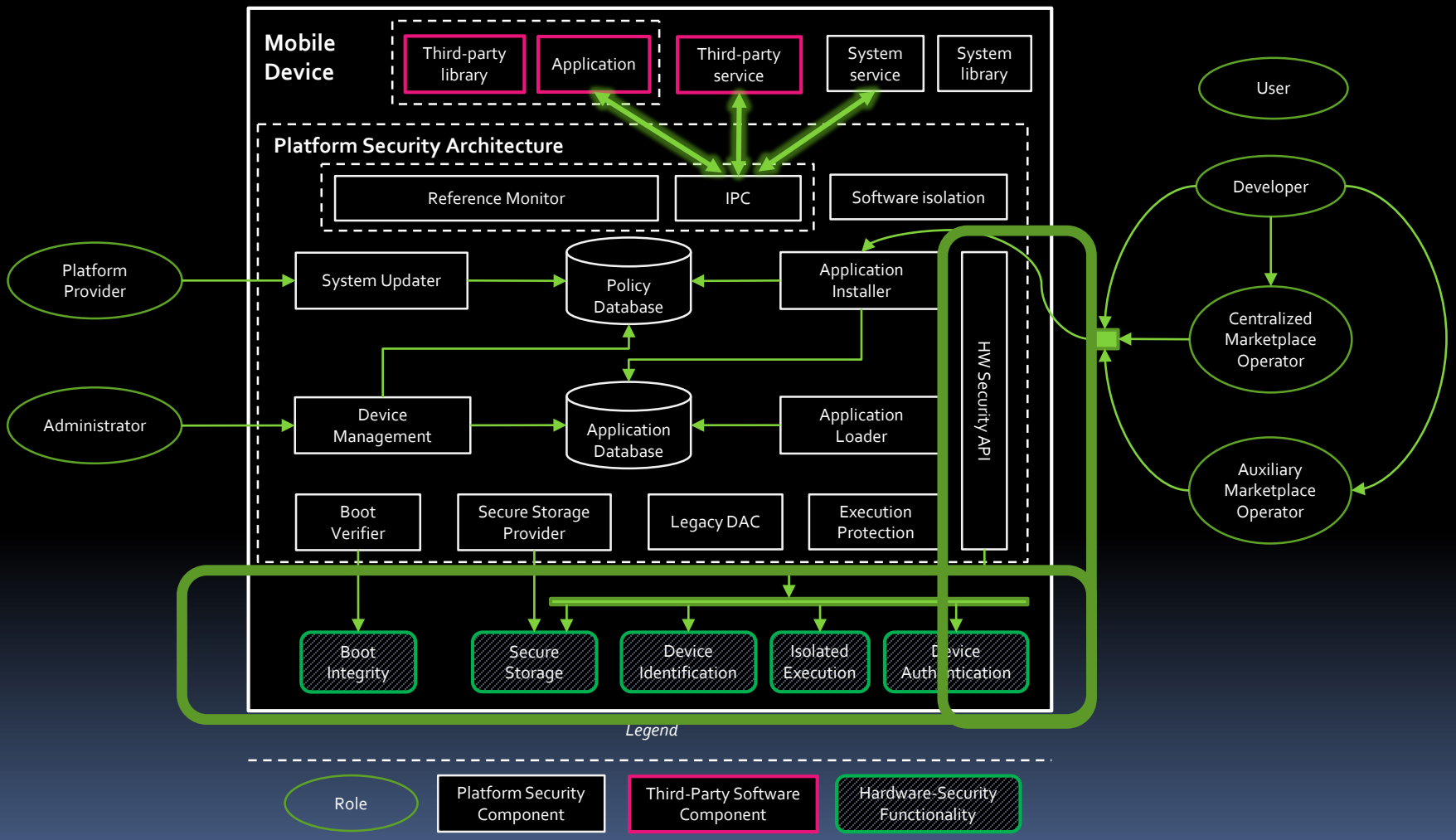
Open issues remain: can you think of some?

Why Generalize?



"The General Problem" http://xkcd.com/974/

Platform security architecture



Hardware platform security

Trusted Execution Environment

HW Security API

What is a TEE?

Processor, memory, storage, peripherals

Trusted Execution Environment

Isolated and integrity-protected

From the “normal” execution environment (Rich Execution Environment)

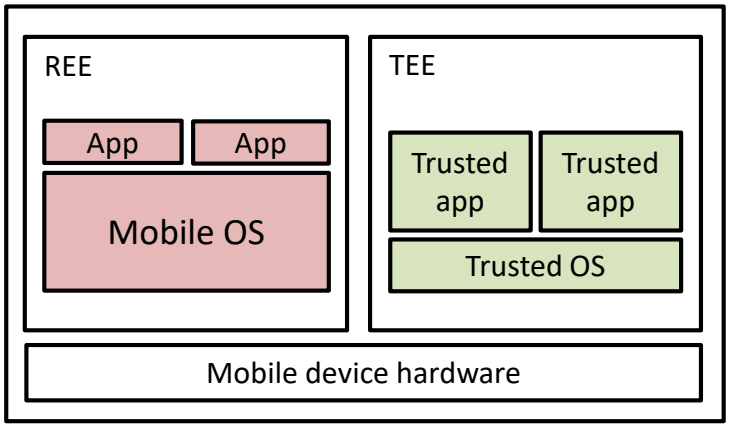
Chances are that:

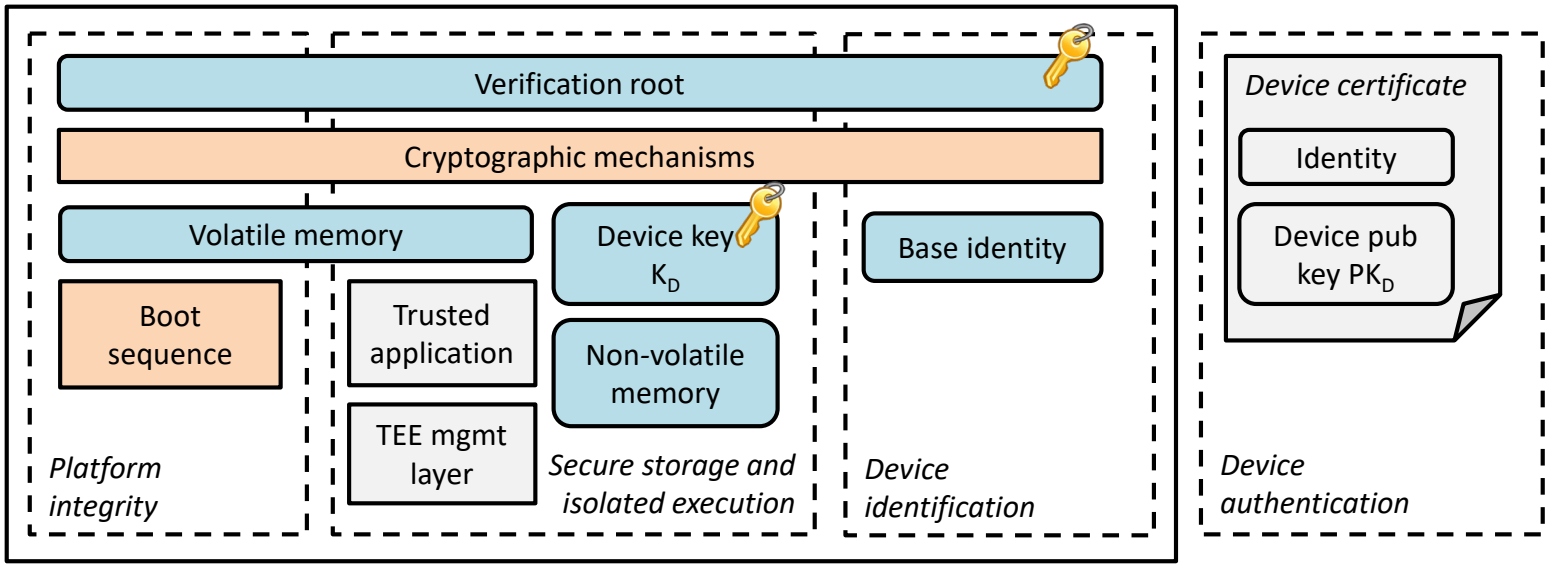
You have devices with hardware-based TEEs in them!

But you don’t have (m)any apps using them

TEE overview

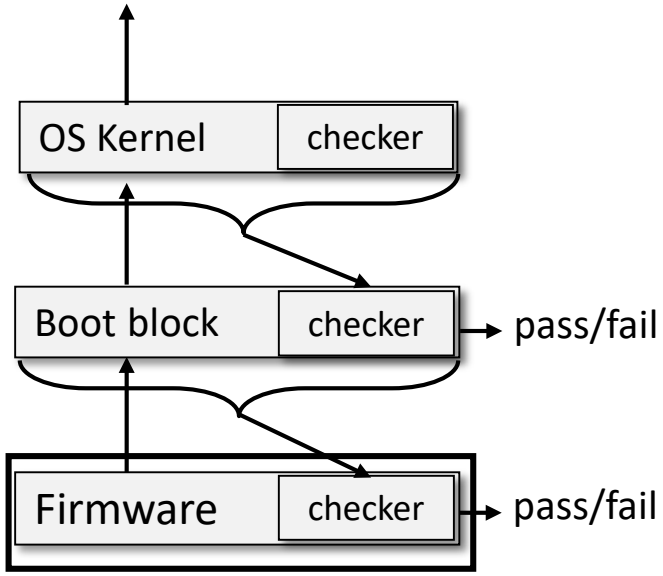
1. Platform integrity (“boot integrity”)
2. Secure storage
3. Isolated execution
4. Device identification
5. Device authentication



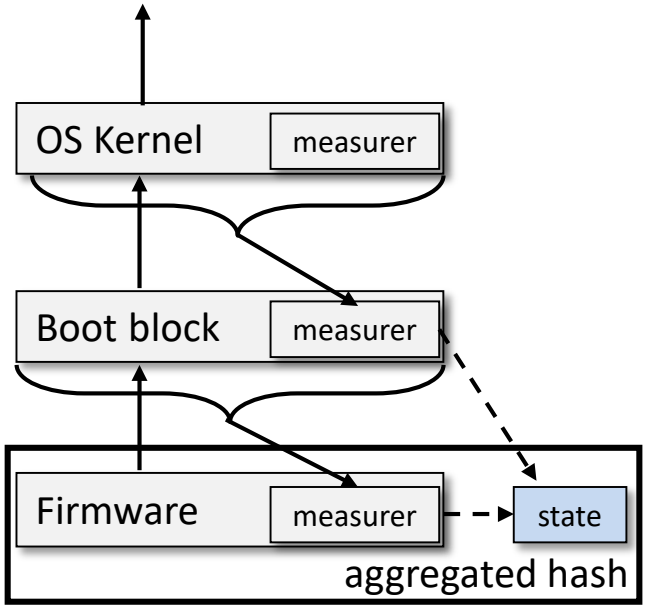


More information in the 2014 IEEE S&P article

Secure boot vs. authenticated boot



Secure boot



Authenticated boot

Why?

How will you implement a checker?

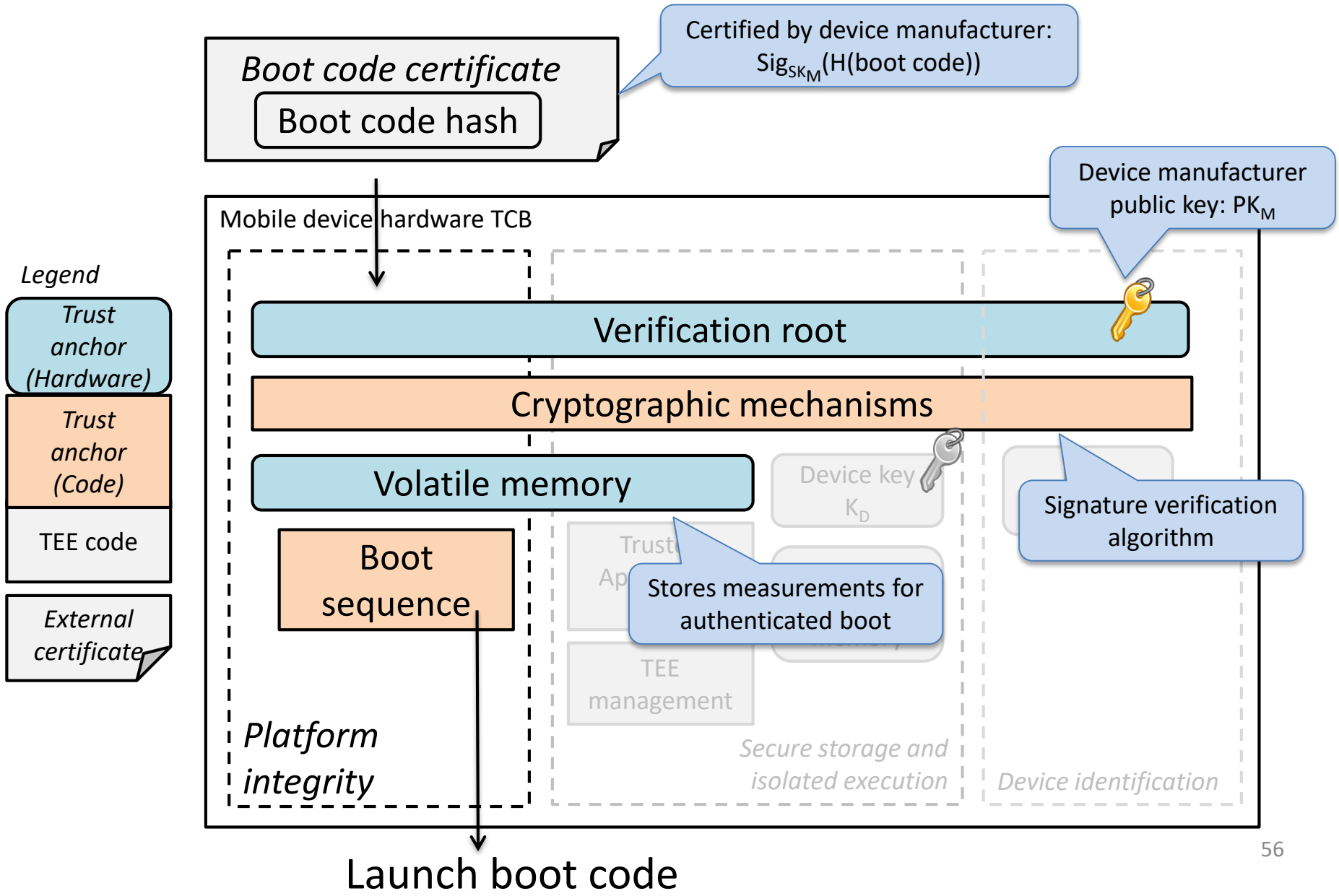
- hardcode H(Boot block|checker) as reference value in checker (in Firmware)?

Why?

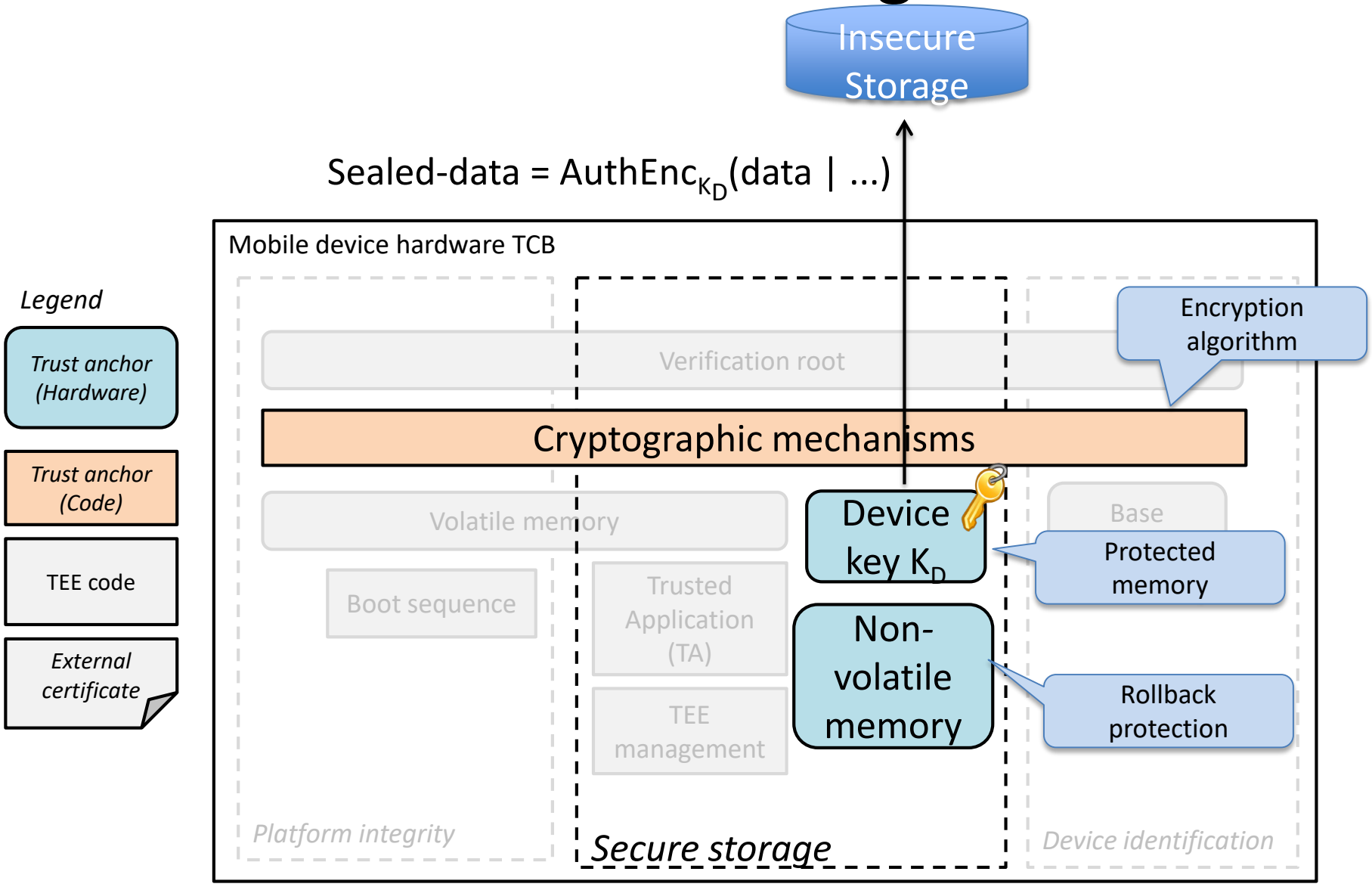
State can be:

- bound to stored secrets (sealing)
- reported to external verifier (remote attestation)

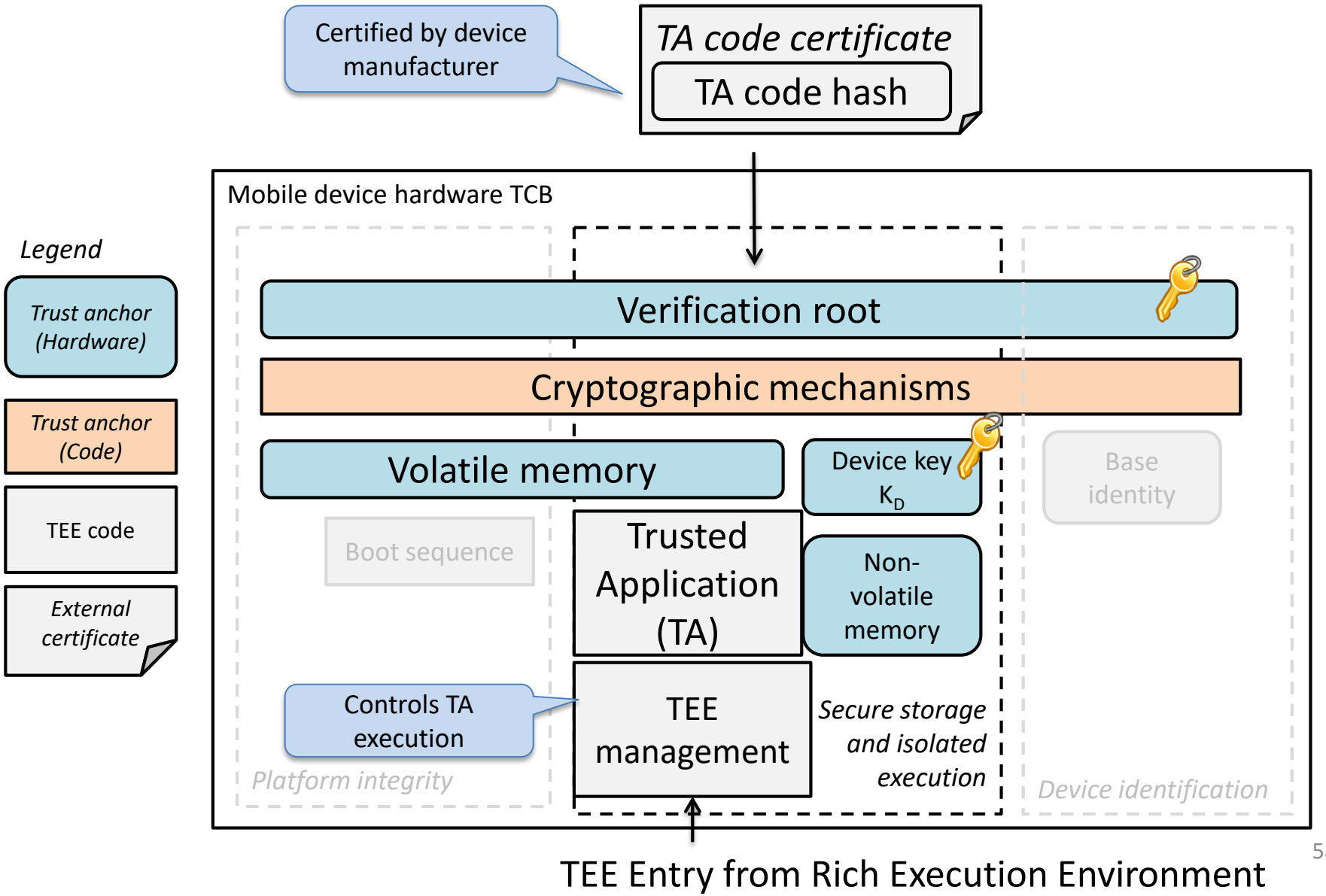
Platform integrity



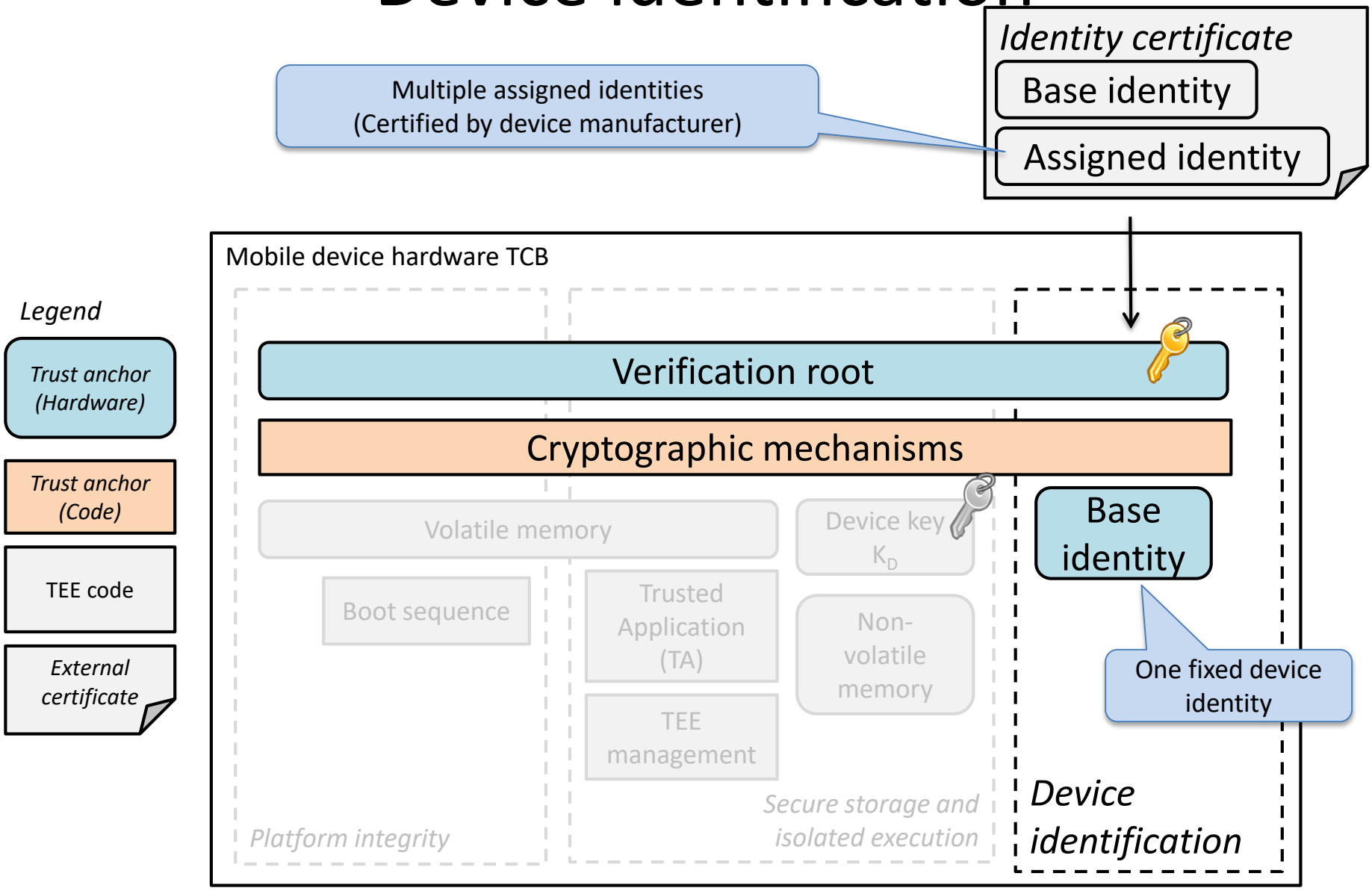
Secure storage



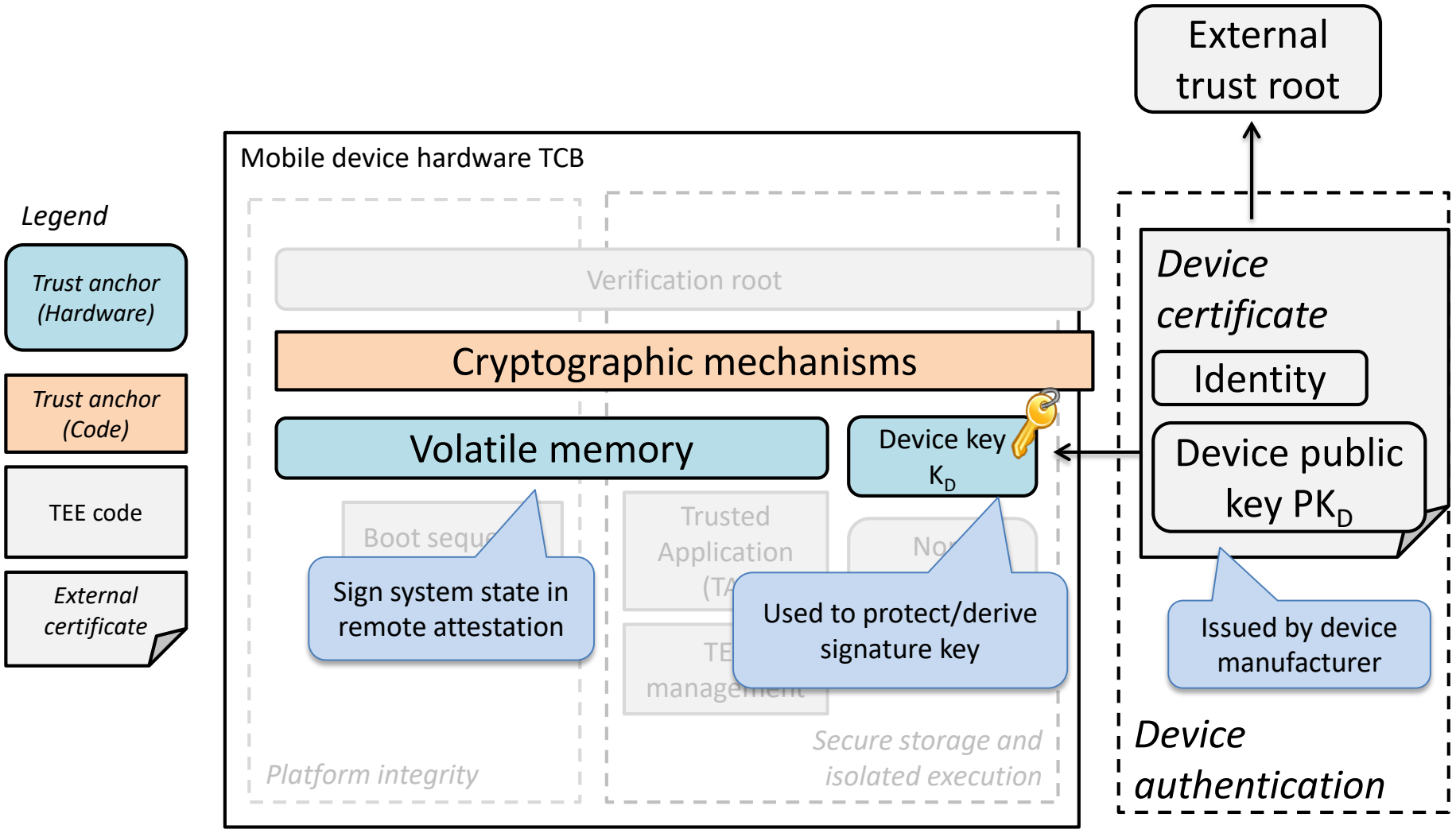
Isolated execution



Device identification

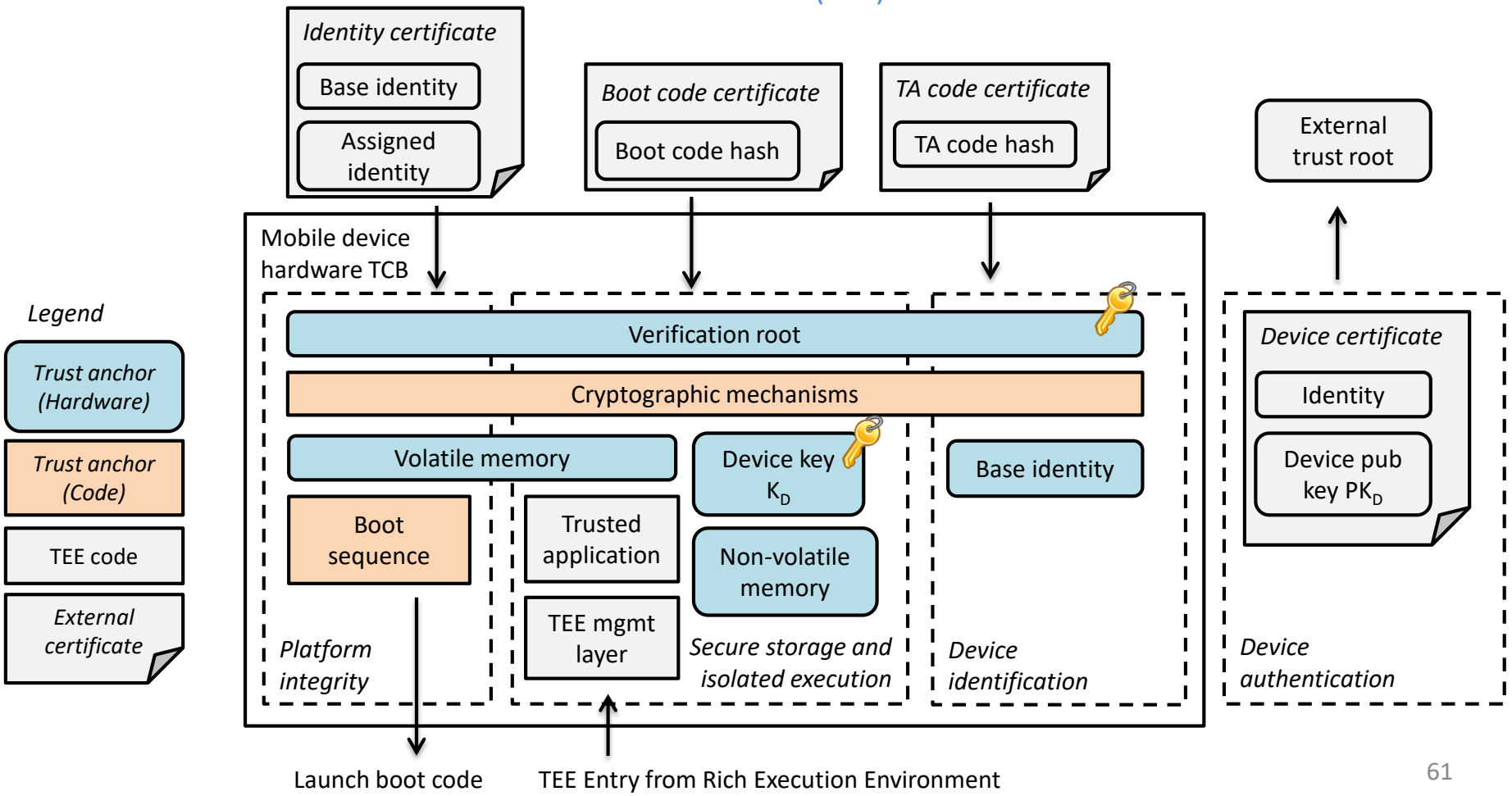


Device authentication (and remote attestation)

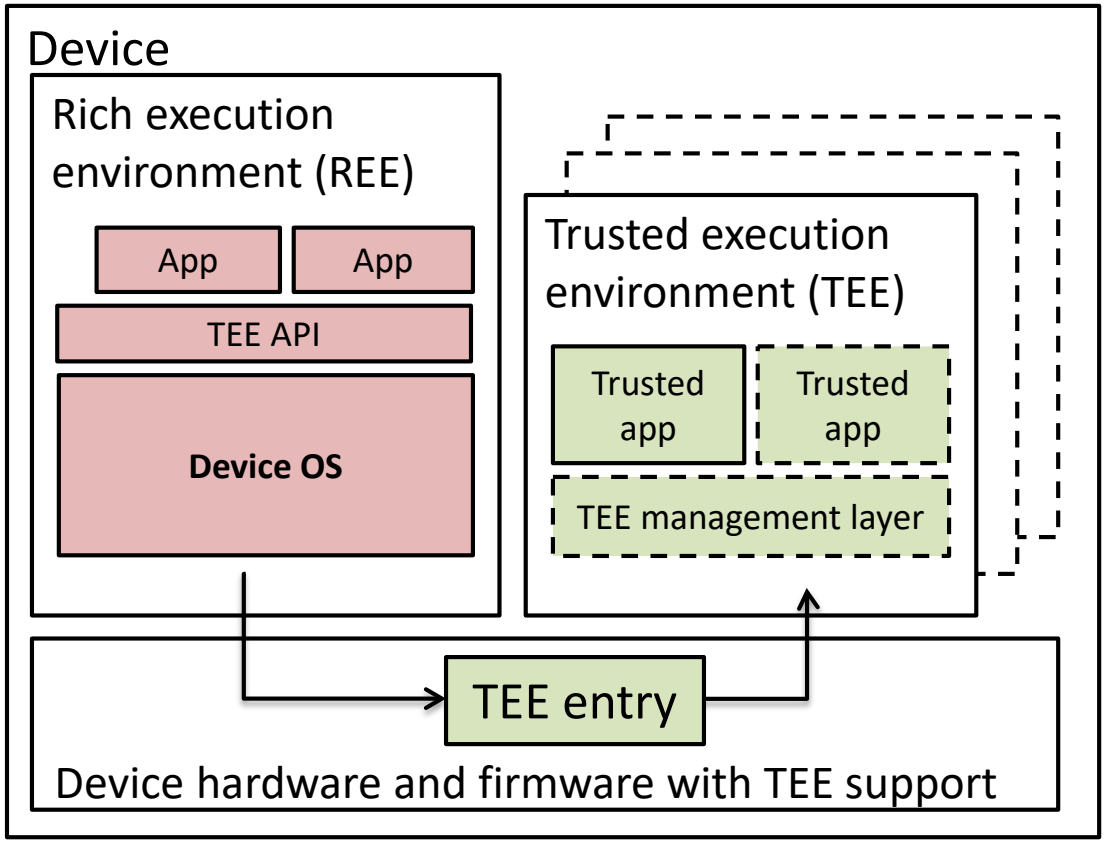


Hardware security mechanisms (recap)

1. Platform integrity
   - Secure boot
   - Authenticated boot
2. Secure storage
3. Isolated execution
   - Trusted Execution Environment (TEE)
4. Device identification
5. Device authentication
   - Remote attestation



TEE system architecture



Architectures with single TEE

- ARM TrustZone
- TI M-Shield
- Smart card
- Crypto co-processor
- Trusted Platform Module (TPM)

Architectures with multiple TEEs

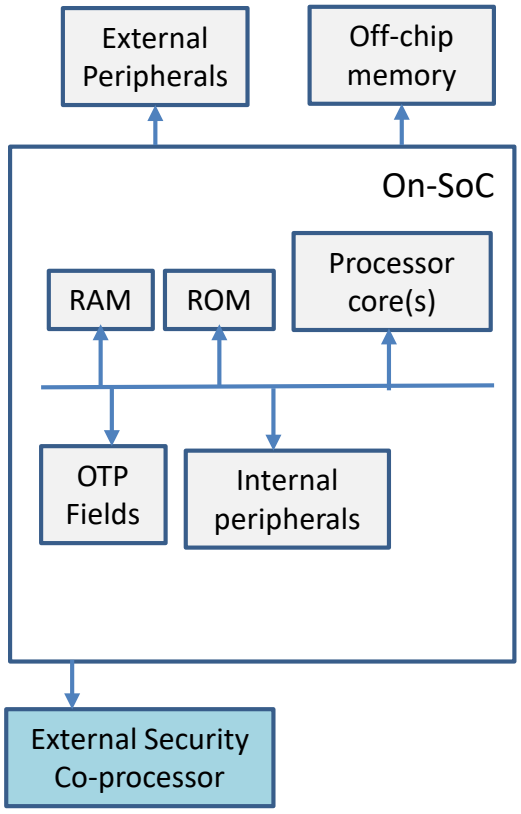
- Intel SGX
- TPM (and "Late Launch")
- Hypervisor

Figure adapted from: Global Platform. TEE system architecture. 2011.

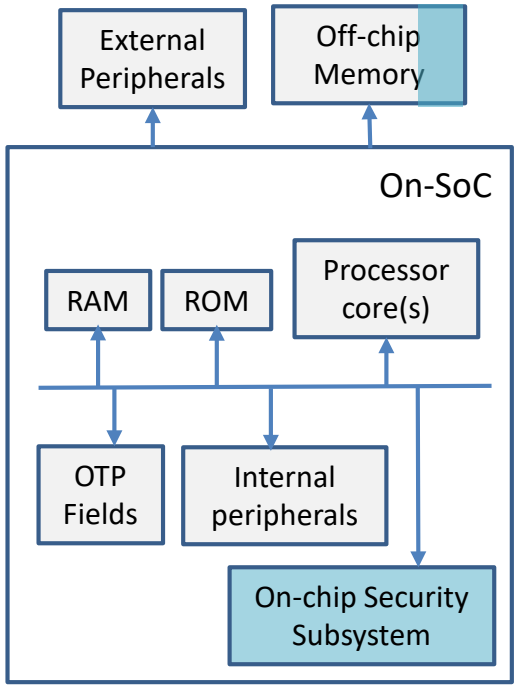
Legend:
SoC : system-on-chip
OTP: one-time programmable

TEE hardware realization alternatives

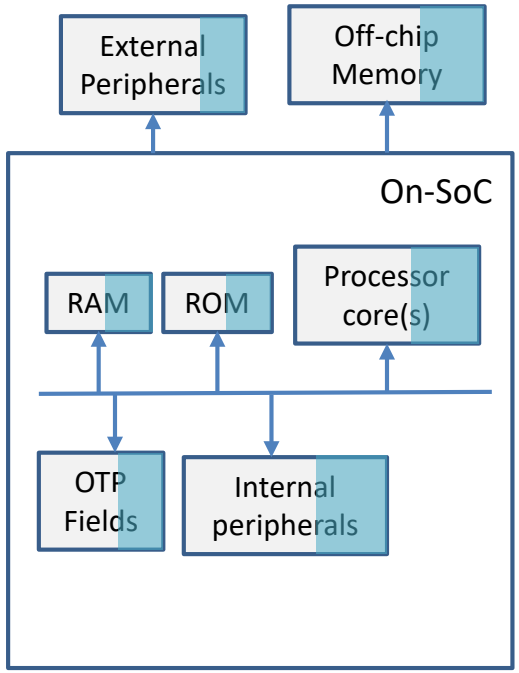
TEE component



External Secure Element
(TPM, smart card)



Embedded Secure Element
(smart card)



Processor Secure Environment
(TrustZone, M-Shield)

Figure adapted from: Global Platform. [TEE system architecture](). 2011.

Did you learn:

- What techniques are used in mobile software platform security?
- What techniques are used in mobile hardware platform security?
- Is there a common general architecture?

Plan for the course

- Lecture 1: Platform security basics
- Lecture 2: Case study – Android OS Platform Security
- Lecture 3: Mobile platform security
- Lecture 4: Hardware security enablers
- Lecture 5: Usability of platform security
- Lecture 6: Summary and outlook
- Lecture 7: SE Android policies
- Lecture 8: Machine learning and security
- Lecture 8: IoT Security