IAIK TU Graz

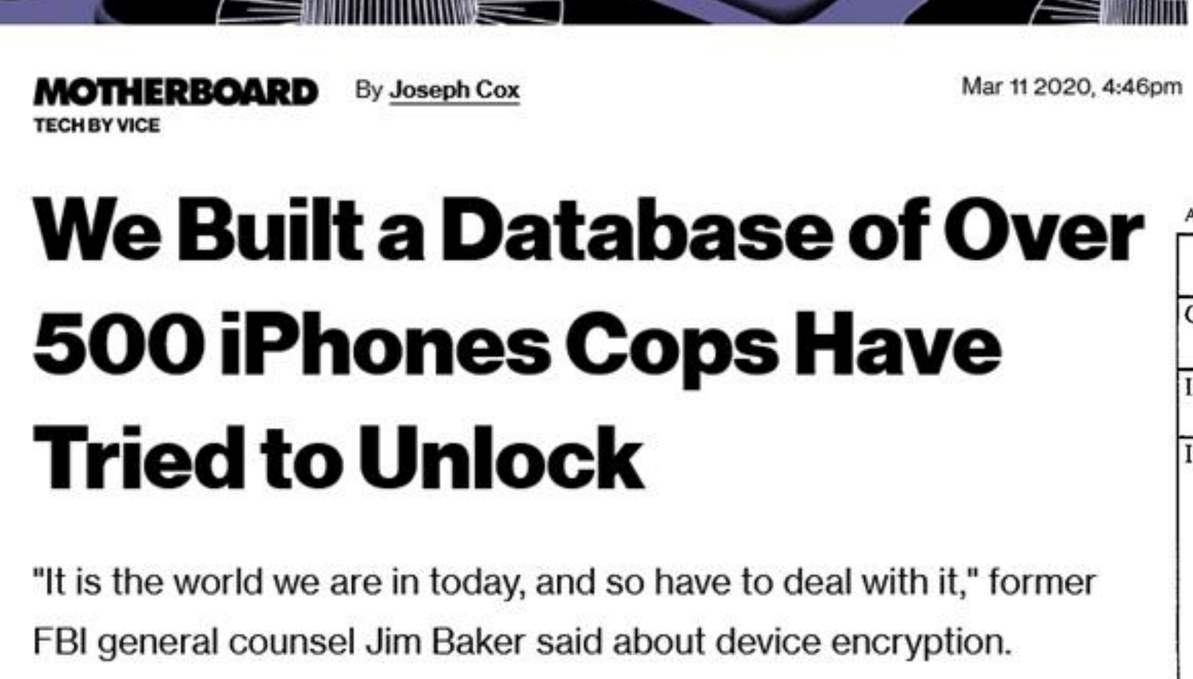
iOS Platform Security

Mobile Security 2021

Johannes Feichtner
johannes.feichtner@iaik.tugraz.at

Outline

- Low-level System Security
- Updates
- Encryption Systems
- Key Management & Passcodes
- Backup

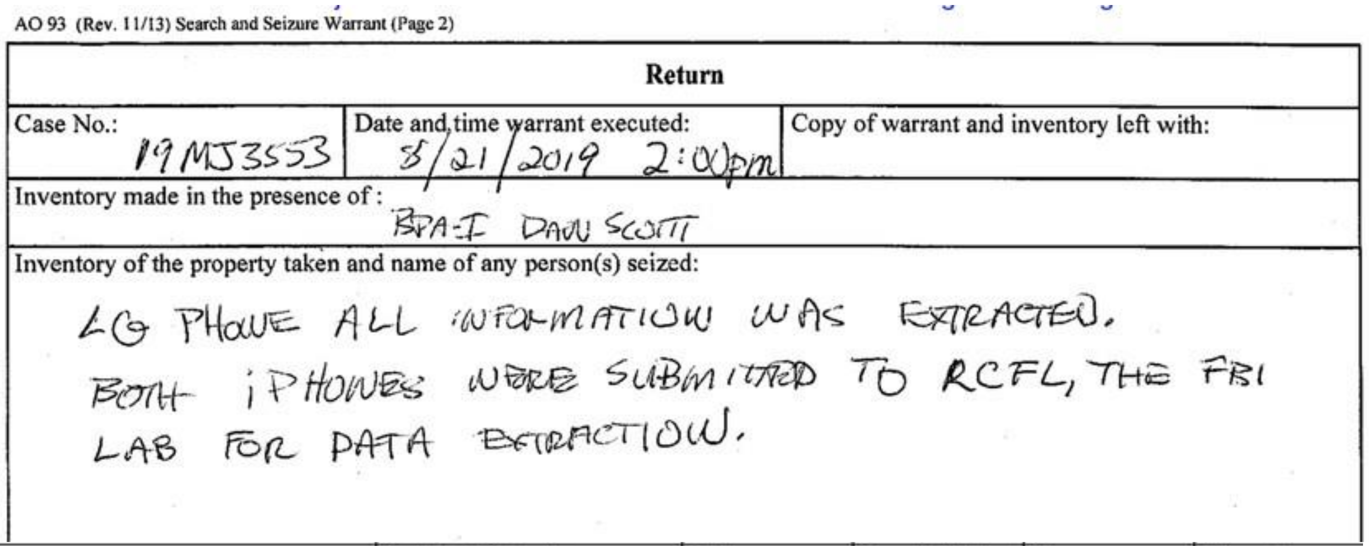


MOTHERBOARD TECH BY VICE By Joseph Cox Mar 11 2020, 4:46pm

See: https://www.vice.com/en_us/article/4ag5yj/unlock-apple-iphone-database-for-police

We Built a Database of Over 500 iPhones Cops Have Tried to Unlock

"It is the world we are in today, and so have to deal with it," former FBI general counsel Jim Baker said about device encryption.



AO 93 (Rev. 11/13) Search and Seizure Warrant (Page 2)

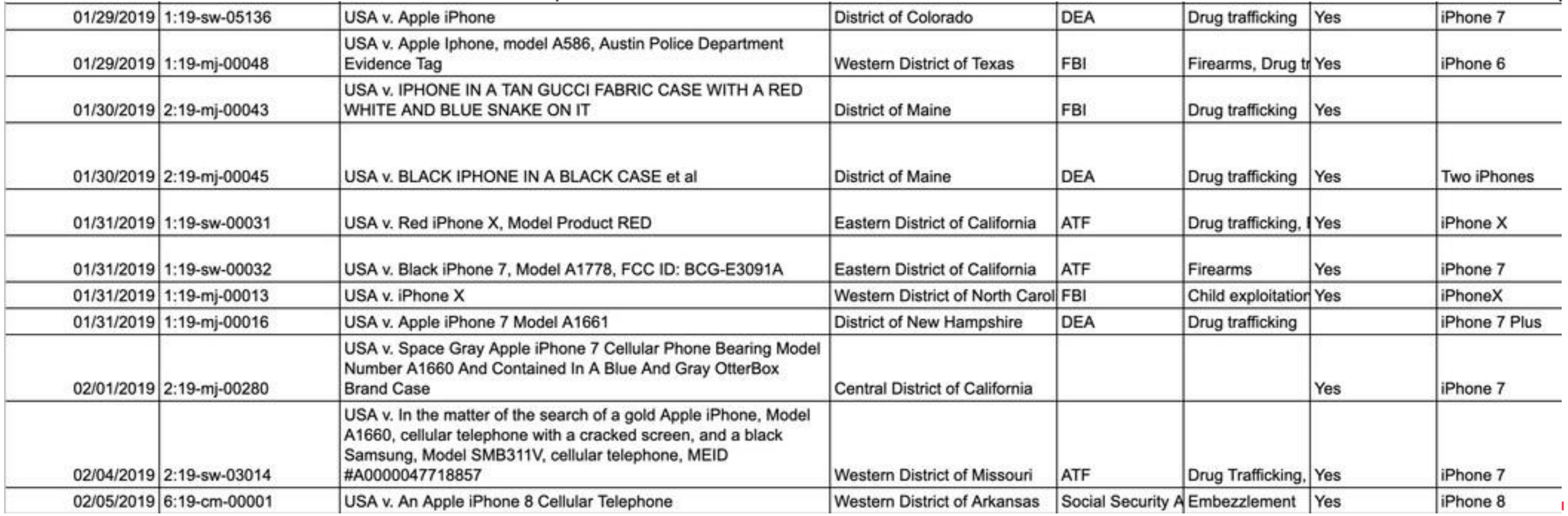
Return

Case No.: 19MJ3553	Date and time warrant executed: 8/21/2019 2:00pm	Copy of warrant and inventory left with:

Inventory made in the presence of : BPA-I DAN SCOTT

Inventory of the property taken and name of any person(s) seized:

LG PHONE ALL INFORMATION WAS EXTRACTED. BOTH iPHONES WERE SUBMITTED TO RCFL, THE FBI LAB FOR DATA EXTRACTION.



01/29/2019	1:19-sw-05136	USA v. Apple iPhone	District of Colorado	DEA	Drug trafficking	Yes	iPhone 7
01/29/2019	1:19-mj-00048	USA v. Apple Iphone, model A586, Austin Police Department Evidence Tag	Western District of Texas	FBI	Firearms, Drug tr	Yes	iPhone 6
01/30/2019	2:19-mj-00043	USA v. IPHONE IN A TAN GUCCI FABRIC CASE WITH A RED WHITE AND BLUE SNAKE ON IT	District of Maine	FBI	Drug trafficking	Yes	
01/30/2019	2:19-mj-00045	USA v. BLACK IPHONE IN A BLACK CASE et al	District of Maine	DEA	Drug trafficking	Yes	Two iPhones
01/31/2019	1:19-sw-00031	USA v. Red iPhone X, Model Product RED	Eastern District of California	ATF	Drug trafficking, I	Yes	iPhone X
01/31/2019	1:19-sw-00032	USA v. Black iPhone 7, Model A1778, FCC ID: BCG-E3091A	Eastern District of California	ATF	Firearms	Yes	iPhone 7
01/31/2019	1:19-mj-00013	USA v. iPhone X	Western District of North Carol	FBI	Child exploitation	Yes	iPhoneX
01/31/2019	1:19-mj-00016	USA v. Apple iPhone 7 Model A1661	District of New Hampshire	DEA	Drug trafficking		iPhone 7 Plus
02/01/2019	2:19-mj-00280	USA v. Space Gray Apple iPhone 7 Cellular Phone Bearing Model Number A1660 And Contained In A Blue And Gray OtterBox Brand Case	Central District of California			Yes	iPhone 7
02/04/2019	2:19-sw-03014	USA v. In the matter of the search of a gold Apple iPhone, Model A1660, cellular telephone with a cracked screen, and a black Samsung, Model SMB311V, cellular telephone, MEID #A0000047718857	Western District of Missouri	ATF	Drug Trafficking,	Yes	iPhone 7
02/05/2019	6:19-cm-00001	USA v. An Apple iPhone 8 Cellular Telephone	Western District of Arkansas	Social Security A	Embezzlement	Yes	iPhone 8

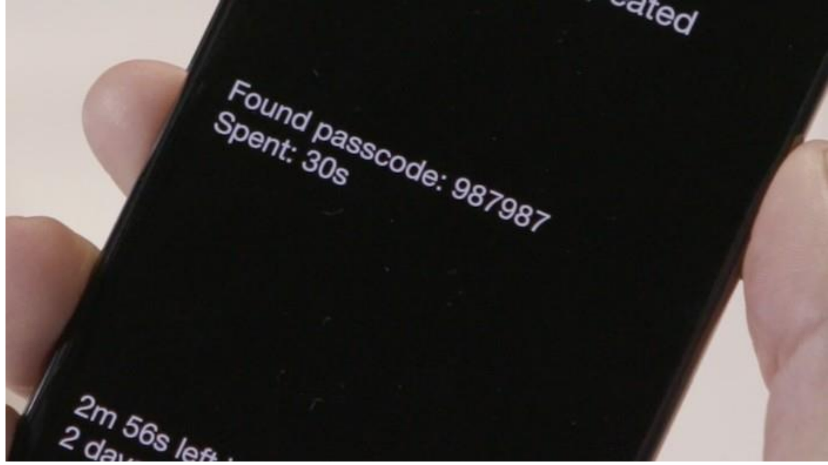
Cops Are Confident iPhone Hackers Have Found a Workaround to Apple's New Security Feature

"Grayshift has gone to great lengths to future proof their technology and stated that they have already defeated this security feature in the beta build."

By **Joseph Cox** and **Lorenzo Franceschi-Bicchierai**

Jun 14 2018, 7:28pm Share Tweet Snap

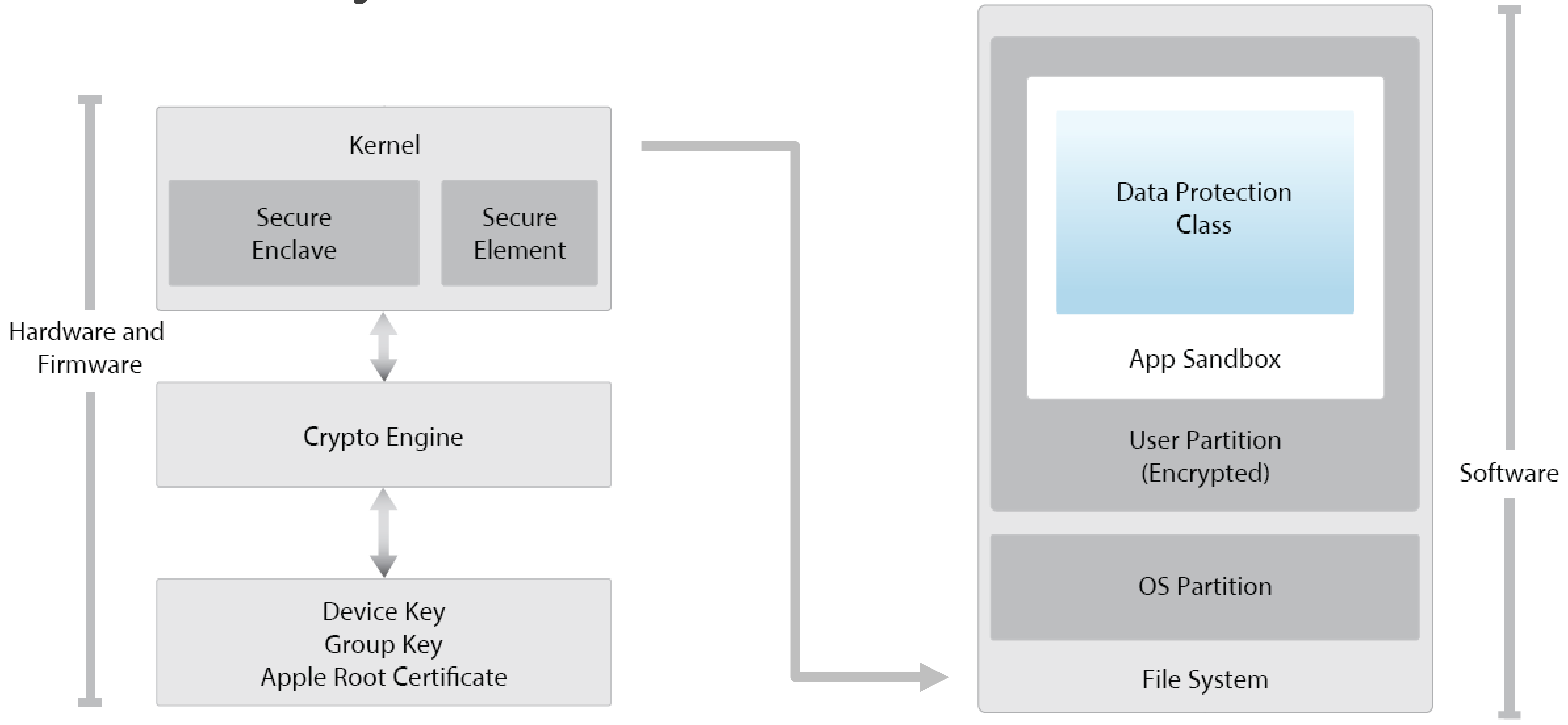
See: https://www.vice.com/en_us/article/pavwzv/cops-are-confident-iphone-hackers-have-found-a-workaround-to-apples-new-security-feature



According to company slides, the device has two strategies to access data on the phone: "Before First Unlock" or BFU, and "After First Unlock" or AFU. BFU is a "slow brute force," meaning it takes 10 minutes per try. This gives access to "limited data." That's likely because the BFU strategy happens when the phone was off when seized. If that's the case, when turned on, the iPhone has most of its data, including contacts, messages and other personal data still encrypted.

AFU, on the other hand is a "fast brute force" mode that presumably kicks in when the phone is locked but was turned on and unlocked at some point by the owner. In this case, it allows for 300,000 tries and allows "parallel extraction of pre-unlock data." If AFU works, the slide adds, "95% of the user's data is available instantly." The slides were shown during a GrayKey presentation at a recent mobile forensics conference in Myrtle Beach, South Carolina.

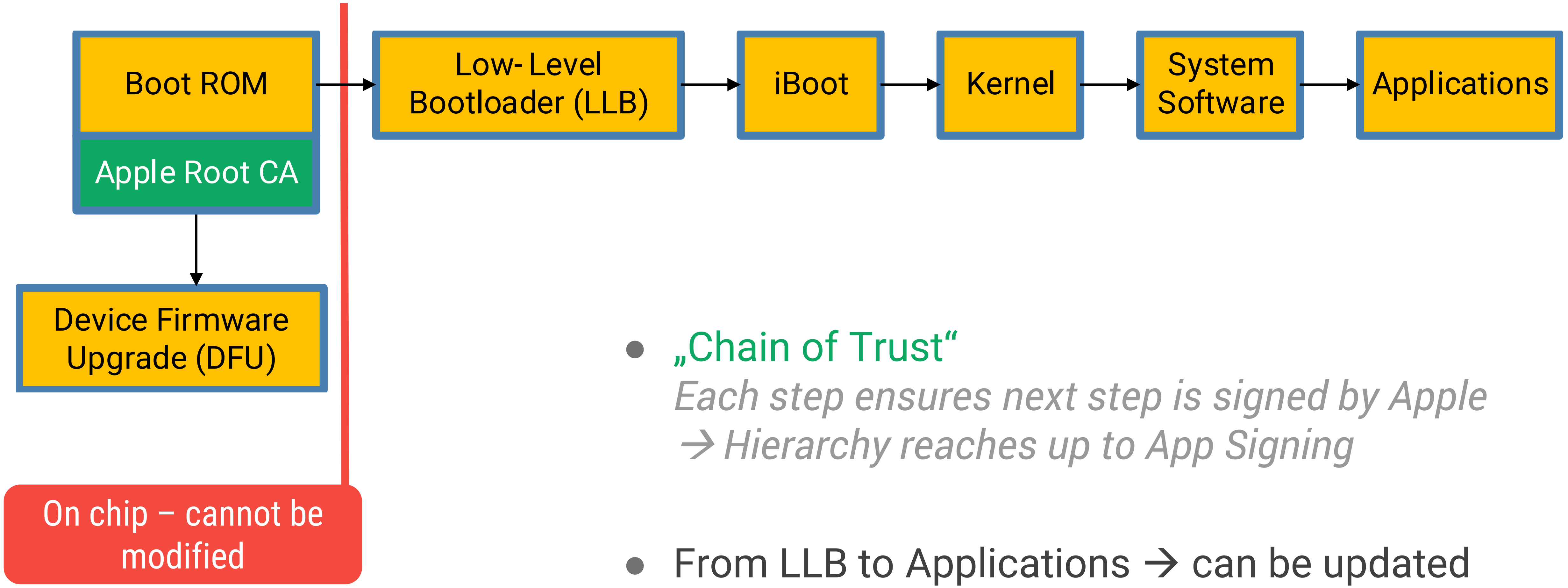
iOS Security Architecture



Source: https://goo.gl/2kl3Qt

Low-Level System Security

Secure Boot Chain



Boot ROM
Apple Root CA
Low- Level Bootloader (LLB)
iBoot
Kernel
System Software
Applications
Device Firmware Upgrade (DFU)
On chip – cannot be modified

- „Chain of Trust“
  Each step ensures next step is signed by Apple
  → Hierarchy reaches up to App Signing

- From LLB to Applications → can be updated

Secure Boot Chain

Starting with simple boot loader…

- Ensure basic security level
- Prevent tampering of lowest software levels

- Similar (separate) boot process for
  - Baseband processor (cellular access)
  - Secure Enclave coprocessor

→ Error if load / verify next step failed
  - Enter DFU (Recovery mode)
  - Connect to iTunes and restore factory defaults

iOS Downgrades?

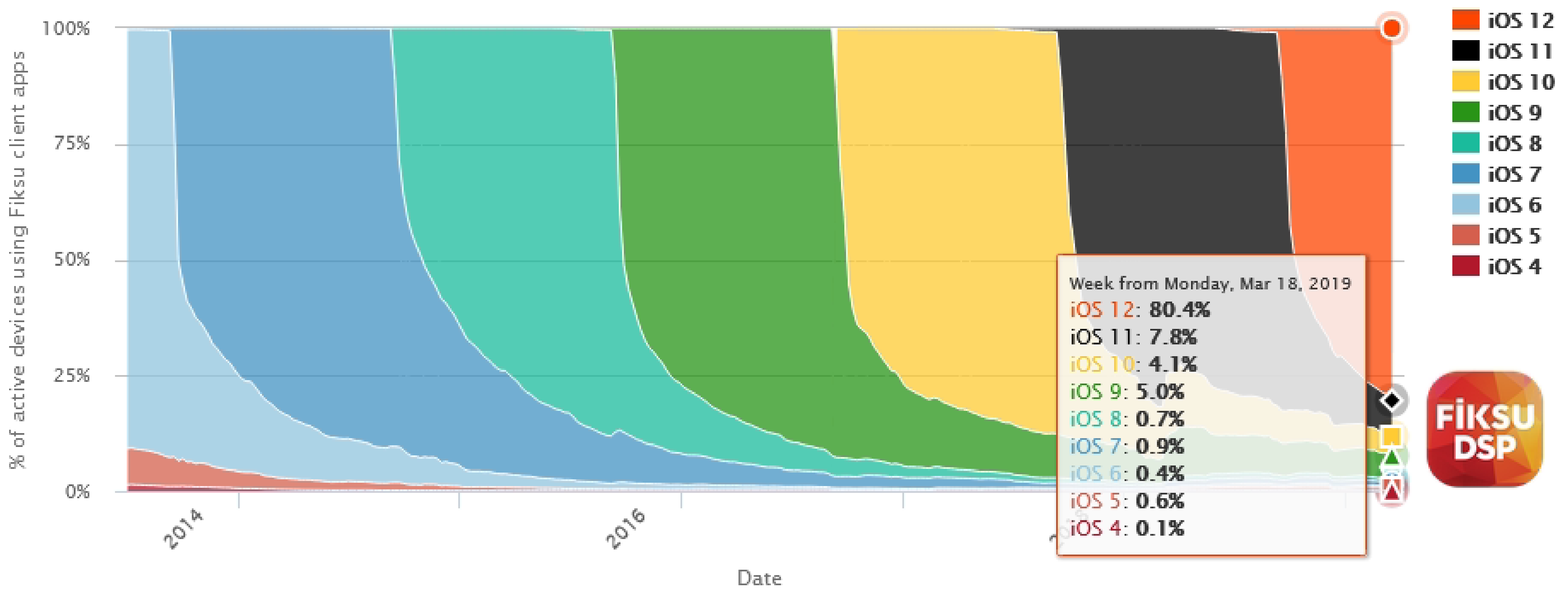
Apple prevents them using „*System Software Authorization*“!

- Signatures alone would enable replay attacks

- Online process
  - Device generates nonce („*anti-replay value*“)
  - Sends unique device ID (UID) + nonce to Apple
  - Apple generates signature for (OS image + Device ID + nonce)
  - Device checks if signature ok, nonce / device ID matches
  - If fine: Install software

- Prevent installation of old OS images by revoking old signatures

iOS Updates



Week from Monday, Mar 18, 2019
iOS 12: 80.4%
iOS 11: 7.8%
iOS 10: 4.1%
iOS 9: 5.0%
iOS 8: 0.7%
iOS 7: 0.9%
iOS 6: 0.4%
iOS 5: 0.6%
iOS 4: 0.1%

Source: https://goo.gl/NJFiDQ

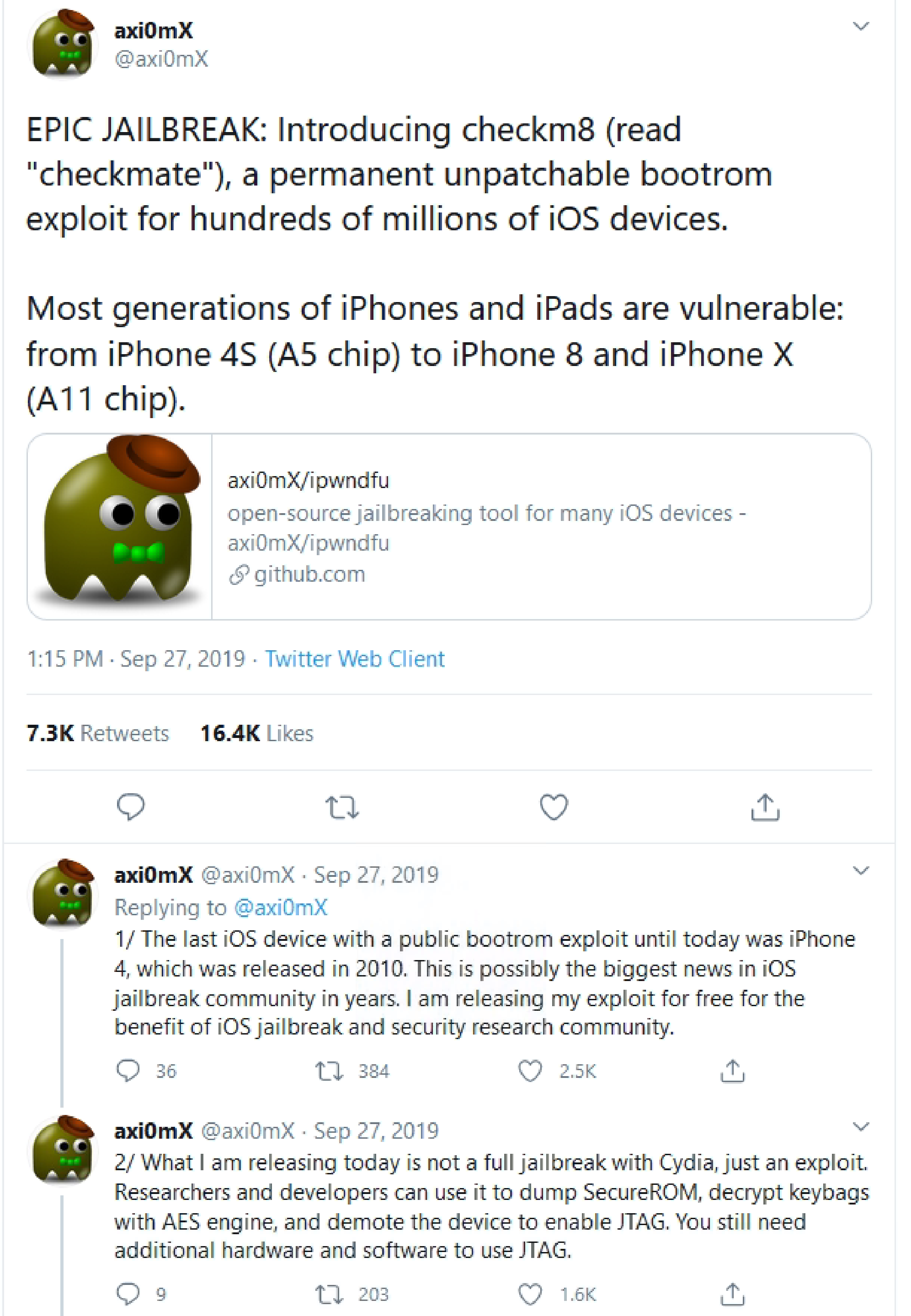
Jailbreak

Available for iOS < 14.4 → fixed on 15.03.2021

- Disable the „Chain of Trust“
  - Kernel modifications
  - Install of unsigned / custom-signed apps

- Hooking point typically: Low-level bootloader
  - Software patchfix possible!

- BootROM exploits
  - Hardware problems → deploy new chips
  - Checkm8 exploit published on 27.9.2019



axi0mX
@axi0mX

EPIC JAILBREAK: Introducing checkm8 (read "checkmate"), a permanent unpatchable bootrom exploit for hundreds of millions of iOS devices.

Most generations of iPhones and iPads are vulnerable: from iPhone 4S (A5 chip) to iPhone 8 and iPhone X (A11 chip).

axi0mX/ipwndfu
open-source jailbreaking tool for many iOS devices - axi0mX/ipwndfu
github.com

1:15 PM · Sep 27, 2019 · Twitter Web Client

7.3K Retweets **16.4K** Likes

axi0mX @axi0mX · Sep 27, 2019
Replying to @axi0mX
1/ The last iOS device with a public bootrom exploit until today was iPhone 4, which was released in 2010. This is possibly the biggest news in iOS jailbreak community in years. I am releasing my exploit for free for the benefit of iOS jailbreak and security research community.
36 384 2.5K

axi0mX @axi0mX · Sep 27, 2019
2/ What I am releasing today is not a full jailbreak with Cydia, just an exploit. Researchers and developers can use it to dump SecureROM, decrypt keybags with AES engine, and demote the device to enable JTAG. You still need additional hardware and software to use JTAG.
9 203 1.6K

Secure Enclave

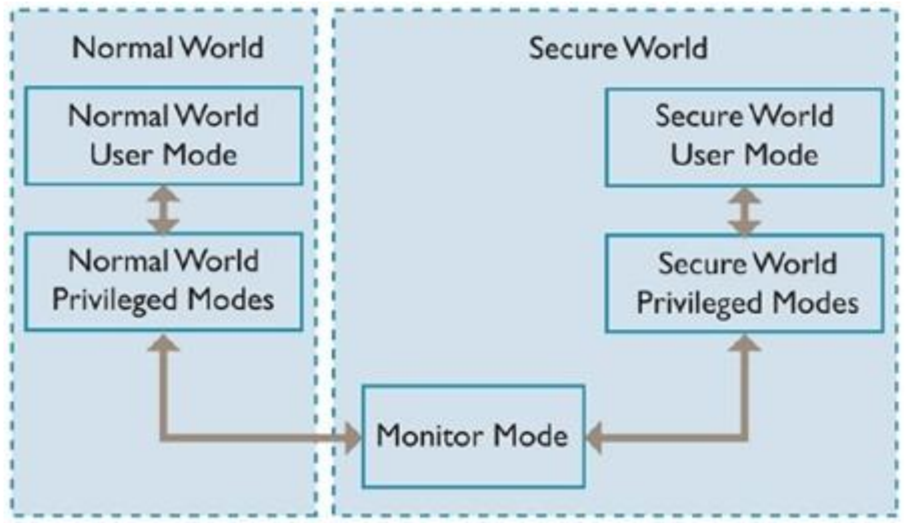
Goals?

- Protect user data by strong cryptographic master key
  - Derived from user passcode
- Prevent…
  - Offline attack on passcode → derive master key in hardware
  - Brute-force attack → hard limit on number of passcode tries
- Hardware keys from master key derivation
  never handed out to mutable software
- Integrated support for alternative unlock mechanisms (FaceID, TouchID, …)

Secure Enclave

Overview

- Crypto operations for *Data Protection* key management
- Maintain integrity of *Data Protection* even if kernel compromised



How?

- Dedicated co-processor for Apple A7 & newer
- Own Boot ROM & software update sequence
- Based on ARM TrustZone + modifications
- Encrypted memory

Secure Enclave

- Dedicated AES-256 crypto engine
- Individual for each coprocessor:
  - Unique device ID (UID)
  - Device group ID (GID)

  $\Big\}$ Fused AES-256 keys
- UID tied to particular device
  - Move the chip to another device → File System Encryption unreadable
  - Used to derive AES keys for data encryption (KeyChain, files, file metadata)
- GID key
  - Common to processor class (e.g. Apple A8), needed for system updates

Secure Enclave

- Not directly readable by firmware, software, JTAG (or other debuggers)
  - Get only encryption / decryption results

Key Generation

- Random Number Generator (CTR_DRBG)
- Entropy from
  - Timing variations during boot
  - Interrupt timing after boot

Key Erasure

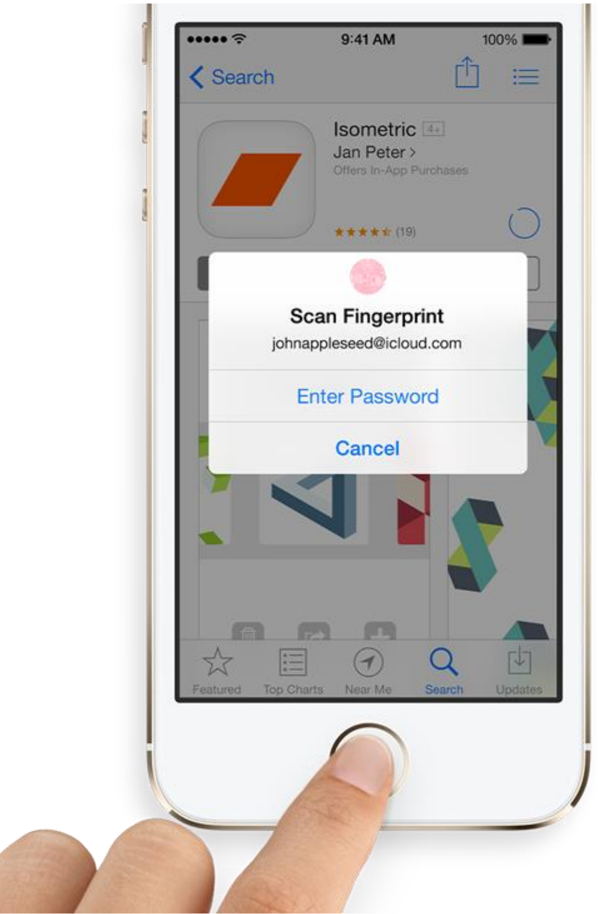
- Address and remove certain blocks on NAND storage
- Triggered with OS option: „Erase all content and settings“

Touch ID

Fingerprint sensing system → in addition to passcode!

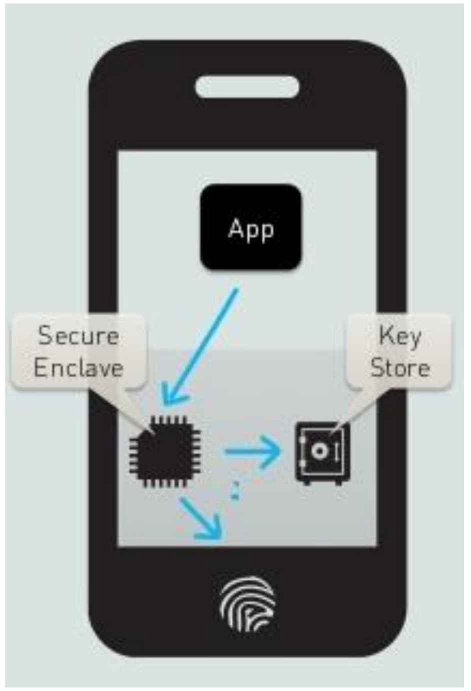
Use to authorize payments (Apple Pay), application access via APIs

5 mismatches possible, then passcode entry required

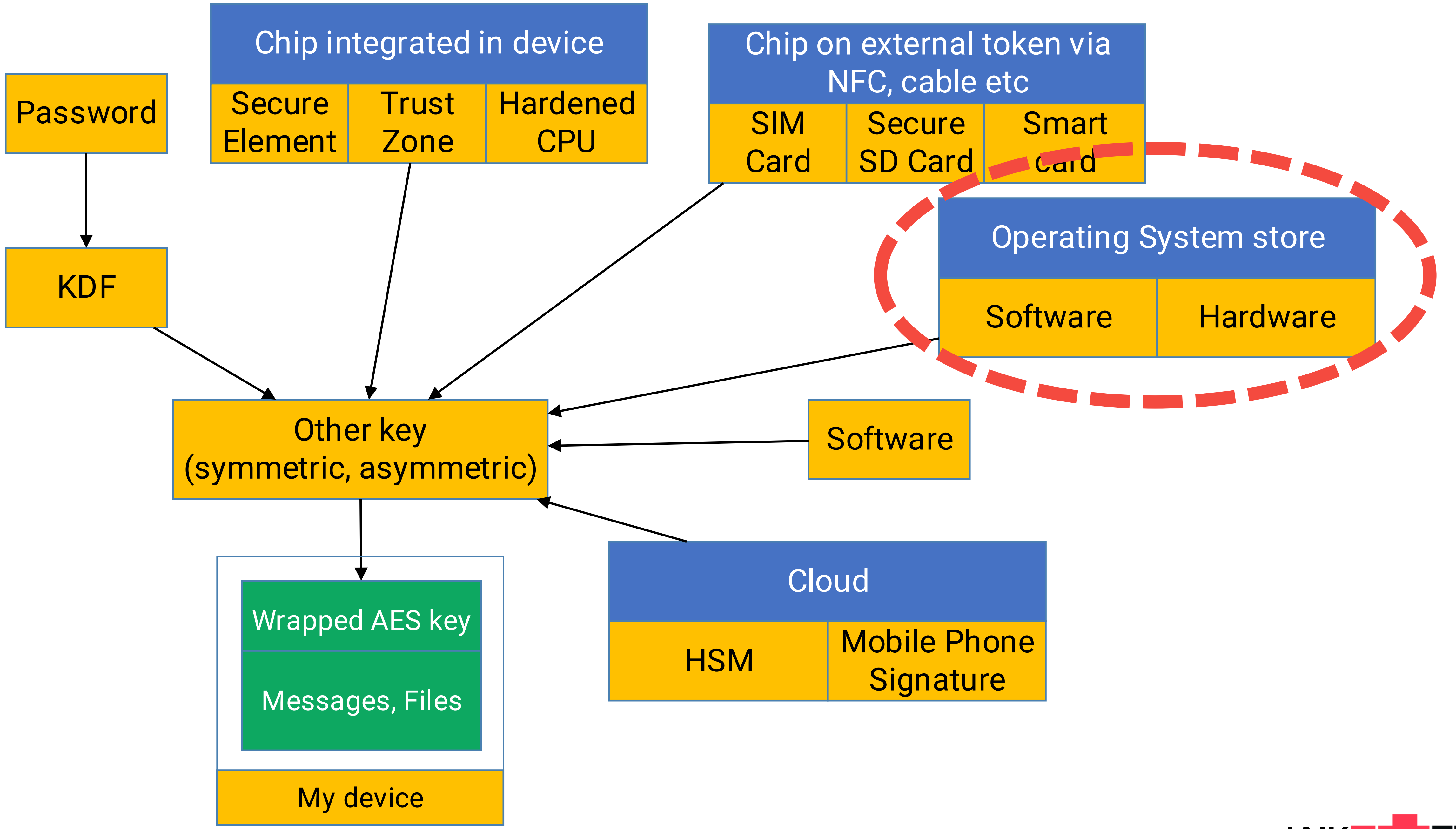


How does device unlock work?

- Without TouchID
  - **Lock**: Data Protection keys discarded
  - **Unlock**: User enters passcode → keys restored
- With TouchID
  - **Lock**: Keys wrapped if TouchID key in Secure Enclave
  - **Unlock**: Fingerprint recognized? → Provide key to unwrap Data Protection keys



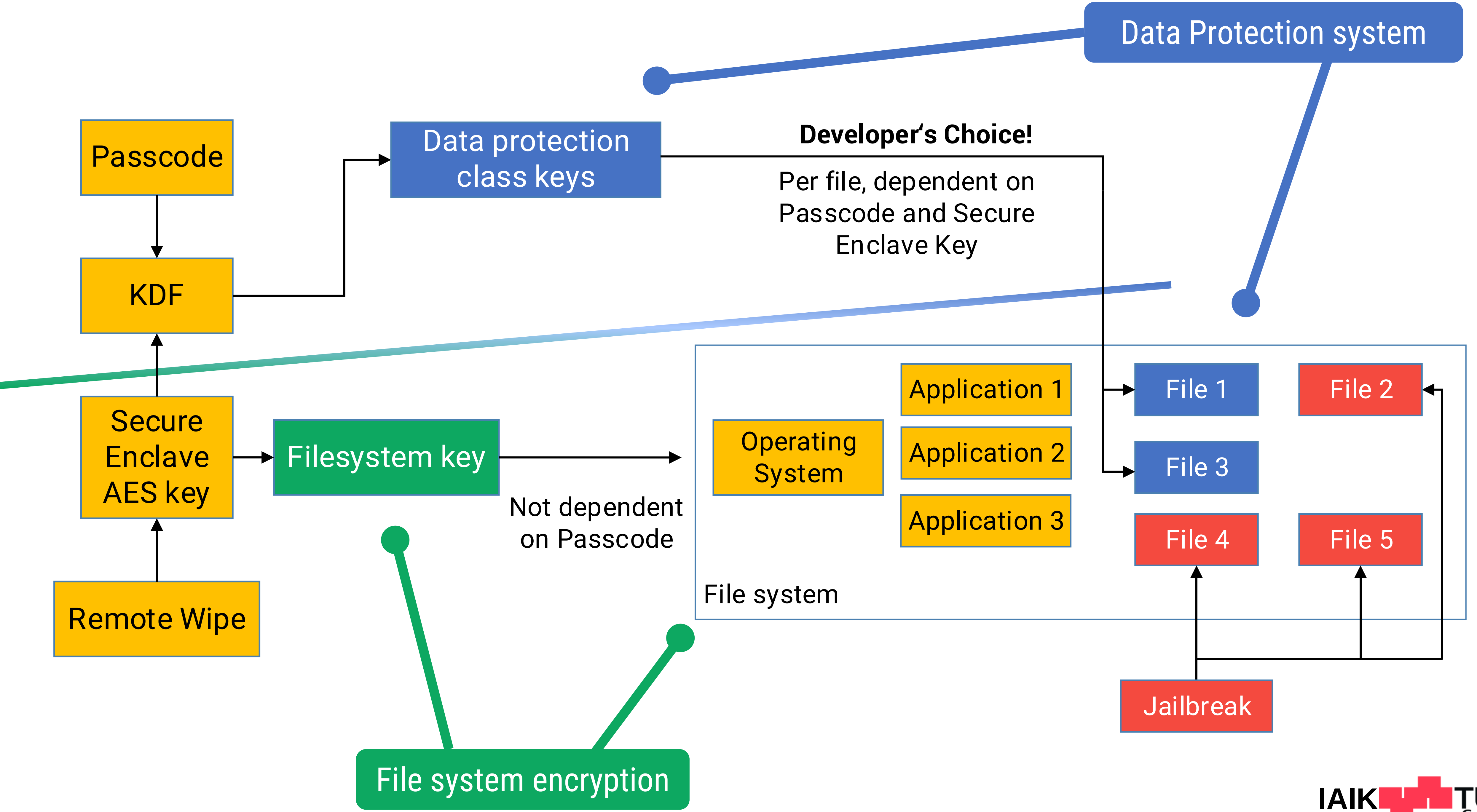
Encryption Systems

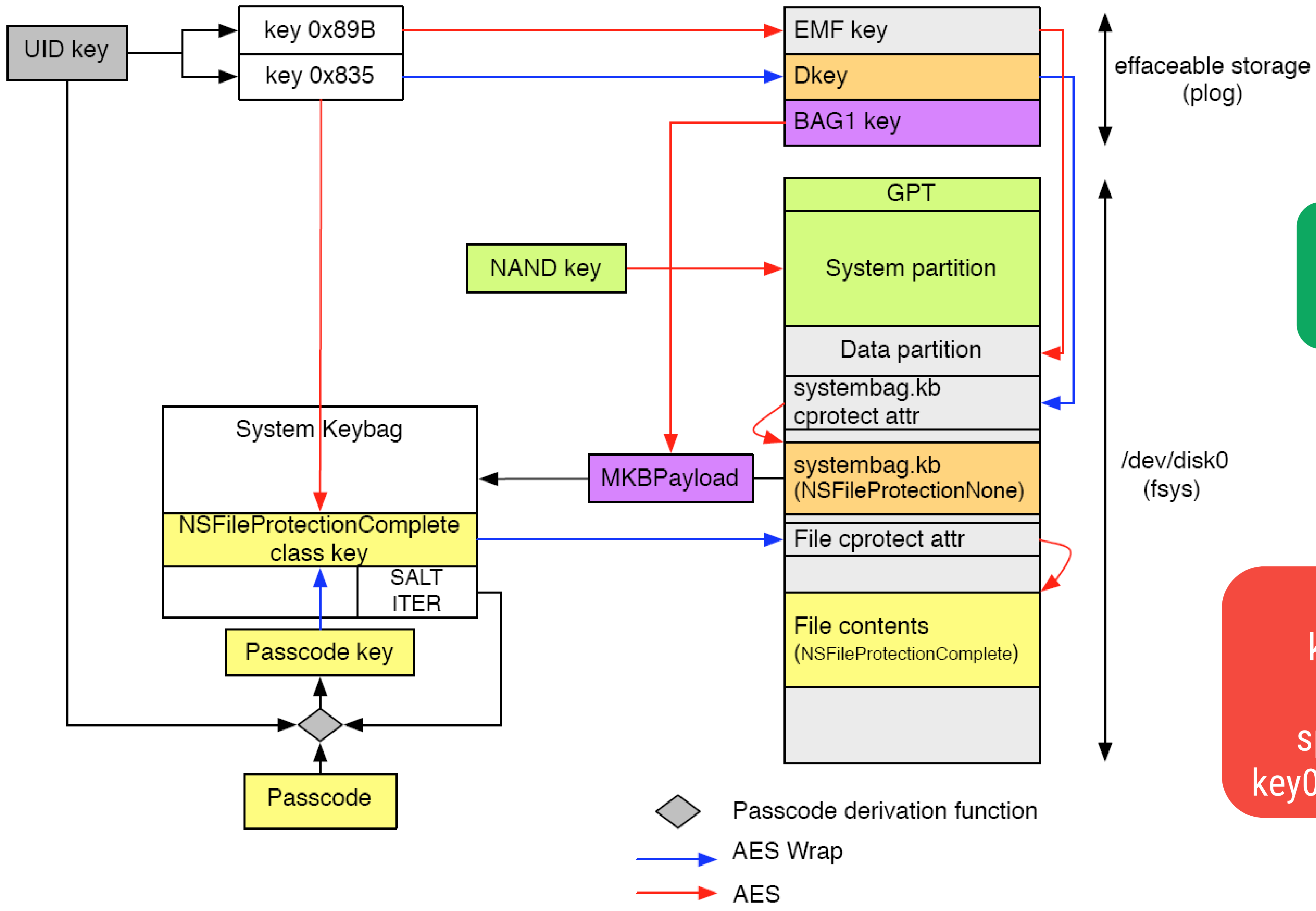

Password
KDF
Chip integrated in device
Secure Element
Trust Zone
Hardened CPU
Chip on external token via NFC, cable etc
SIM Card
Secure SD Card
Smart card
Operating System store
Software
Hardware
Other key (symmetric, asymmetric)
Software
Wrapped AES key
Messages, Files
My device
Cloud
HSM
Mobile Phone Signature

Encryption Systems

- File system encryption
  - Alias: „Full disk encryption“, „Storage encryption“
  - Introduced with iOS 3 and iPhone 3GS
  - Based on hardware element

- Data Protection
  - Introduced with iOS 4
  - Extends File system encryption
  - Improved in newer version (new Protection classes, KeyChain features)


Data Protection system
Passcode
Data protection class keys
Developer‘s Choice!
Per file, dependent on Passcode and Secure Enclave Key
KDF
Secure Enclave AES key
Filesystem key
Not dependent on Passcode
Operating System
Application 1
Application 2
Application 3
File 1
File 2
File 3
File 4
File 5
File system
Remote Wipe
Jailbreak
File system encryption


UID key
key 0x89B
key 0x835
EMF key
Dkey
BAG1 key
effaceable storage (plog)
GPT
System partition
NAND key
Data partition
systembag.kb cprotect attr
System Keybag
MKBPayload
systembag.kb (NSFileProtectionNone)
/dev/disk0 (fsys)
NSFileProtectionComplete class key
File cprotect attr
SALT
ITER
Passcode key
File contents (NSFileProtectionComplete)
Passcode
Passcode derivation function
AES Wrap
AES

Outline
Same as last slide but more detailed!

Note
key0x9B, key0x835
 → Static device-specific nonces, e.g. key0x9B = AES(UID, const)

File System Encryption

Every Secure Enclave Processor has access to a unique private key = UID

- UID key generated by Secure Enclave **itself** immediately after fabrication
- Used to derive / protect further user keys
  - Needed to encrypt the filesystem, files, file metadata
    - User keys are stored on the device but encrypted with the UID key
    - PIN / Passcode is not used for filesystem key derivation, only the UID key

What if phone is stolen?

Apply jailbreak to bypass passcode protection, system decrypts the data for you

→ Makes sense only for quick remote wiping

File System Encryption – Remote Wipe

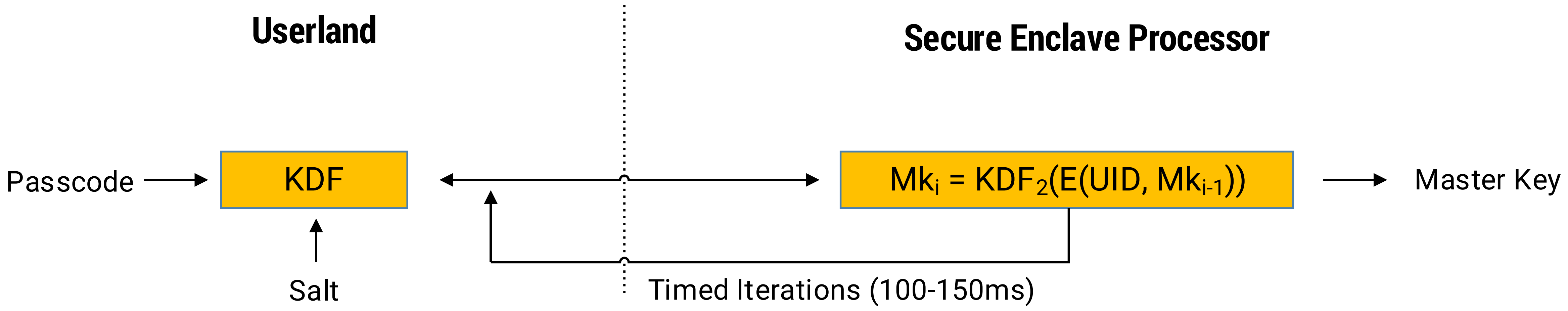
From the iOS Security Guide (Q4 / 2020):

```
The metadata of all files in the file system is encrypted with a random key, which
is created when iOS is first installed or when the device is wiped by a user. The
file system key is stored in Effaceable Storage. Since it’s stored on the device,
this key is not used to maintain the confidentiality of data; instead, it’s
designed to be quickly erased on demand (by the user, with the “Erase all content
and settings” option, or by a user or administrator issuing a remote wipe command
from a mobile device management (MDM) server, Exchange ActiveSync, or iCloud).
Erasing the key in this manner renders all files cryptographically inaccessible.
```

→ Erase the file system key to avoid further access to any file!

→ Remote Wipe does not actually *delete* the file…

Master Key Derivation


Userland
Secure Enclave Processor
Passcode
KDF
Salt
$Mk_i = KDF_2(E(UID, Mk_{i-1}))$
Master Key
Timed Iterations (100-150ms)

- User keys are *wrapped* by master key
- Master key derived from
  - User passcode, random salt + Key derivation function **in** Secure Enclave

User Keybags

Idea

- Manages keys for file and KeyChain protection classes
  - Set of keys generated for each system user
- 10 incorrect tries → further attempts blocked

Different policy associated with each keybag key (*usage, availability*)

- Class A (256-bit AES) → Key only available while device unlocked
- Class B (Curve 25519) → Public key always available, private only while unlocked
- Class C (256-bit AES) → Available after first unlock
- Class D (256-bit AES) → Always available

Data Protection

Status quo

File system encryption encrypts all file blocks with AES-XTS with 128-bit keys

Data protection = Additional encryption layer

- Encrypts each file on user partition separately if device locked or powered off
- Encrypted data protected by user‘s passcode
  - Limit speed of bruteforce attacks with custom passcode derivation function

Design

- Data availability
  - when unlocked, while locked, after first unlock, always
- Protection class for every single file and KeyChain item

 Unique key for every file!

Data Protection Classes

Represent the policies to determine when data is accessible

File Availability	**File Data Protection**
When unlocked	`NSFileProtectionComplete`
While locked	`NSFileProtectionCompleteUnlessOpen`
After first unlock	`NSFileProtectionCompleteUntilFirstUserAuthentication`
Always	`NSFileProtectionNone`

- „Always“: File keys protected with (= „wrapped by“) Class encryption keys only
  → no real protection!
- All others: File keys encrypted with key derived from UID and passcode
  → Jailbreak does not reveal the encrypted data

Data Protection Classes

- `NSFileProtectionComplete`
  - Keys removed from memory when device locked
    → files not available in locked state

- `NSFileProtectionCompleteUnlessOpen`
  - Problem: Some files need to be written when locked, e.g. incoming mails
  - Solution: Use ephemeral ECC keys with derived symmetric keys
  - Private key not available when locked!

- `NSFileProtectionCompleteUntilFirstUserAuthentication`
  - Key available after first unlock (almost like `NSFileProtectionComplete`)

Data Protection – How it works

New file is created…

- Data Protection creates new 256-bit ***per file*** key
- Hardware AES engine uses key to encrypt file content
- ***Per file*** key wrapped with one of several class keys
- Resulting wrapped key stored in file metadata

AES Key Wrap Algorithm (RFC 3394)

File is opened…

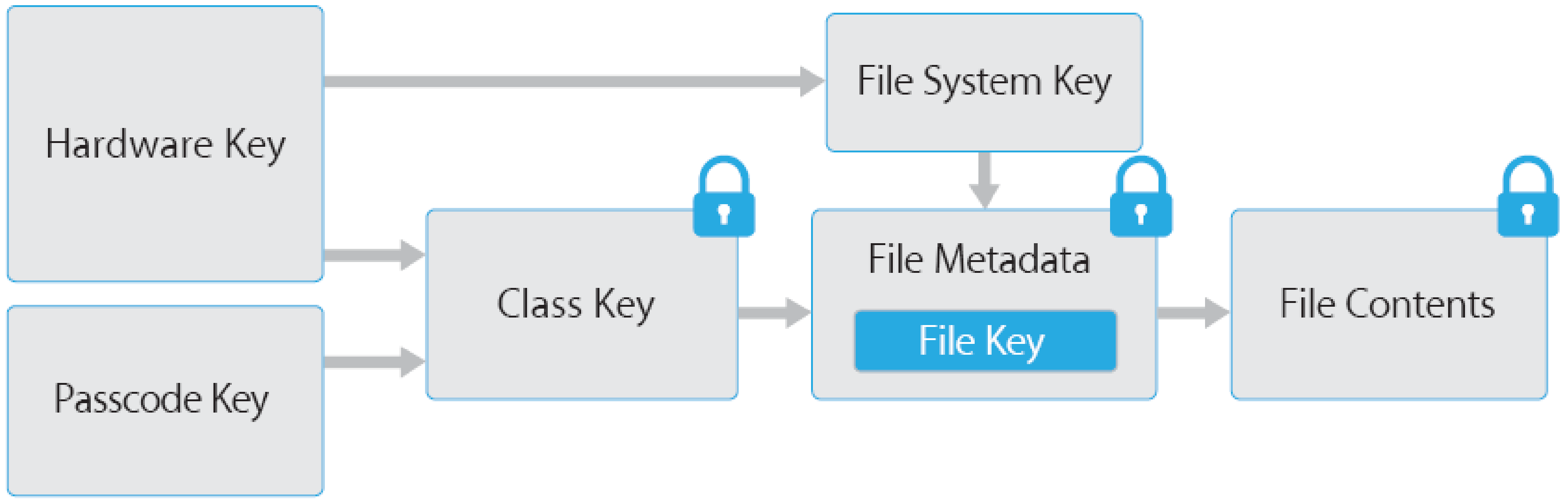
- Metadata decrypted with file system key
  - Result: Wrapped file key and class used for protection
- Unwrap file key with according class key
- Suppy file and key to AES engine & read plaintext result

Note: Wrapped file key handling <u>always</u> in SE! Never enters the CPU

Data Protection

Change file class? Just rewrap file key!

Change passcode? Just rewrap class key!



Hint: To keep it simple… read from right to left ;)

Data Protection – Where is the problem?

- Every new file gets assigned a protection class by an app (!)
  - Handled by the developer!
  - User cannot know which apps encrypt their data and which do not

- Consider the scenario
  - Getting email with PDF attachment (mail app uses data protection)
  - Opening the mail in a PDF reader (not using data protection)

How to find out? → Application Analysis

- Dynamic approach: Monitor live file access using jailbroken device
- Static approach: Look for file API calls + parameters in binary dump

Data Protection – In Practice

```
let fileManager = FileManager.default
fileManager.createDirectory(atPath: folder.path, withIntermediateDirectories: true,
attributes: [FileAttributeKey.protectionKey: FileProtectionType.complete])
…
fileManager.createFile(atPath: databaseKeyURL.path, contents: nil,
attributes: [FileAttributeKey.protectionKey: FileProtectionType.complete])
```

```
let data = Data(count: count)
data.write(to: fullCachePath,
options: [.atomic, .completeFileProtection])
```

Since iOS 7 default protection class: *„Protected until first user authentication“*

Key Management & Passcodes

iOS KeyChain

What for?

Mobile OS needs to handle passwords, login tokens, PINs, certificates, etc

How does it look like?

- 1 SQLite database stored on file system
- KeyChain entries can be shared between apps from same developer *(app group)*
- Access from apps using ordinary API
- Protection classes similar to those for files

Side note:

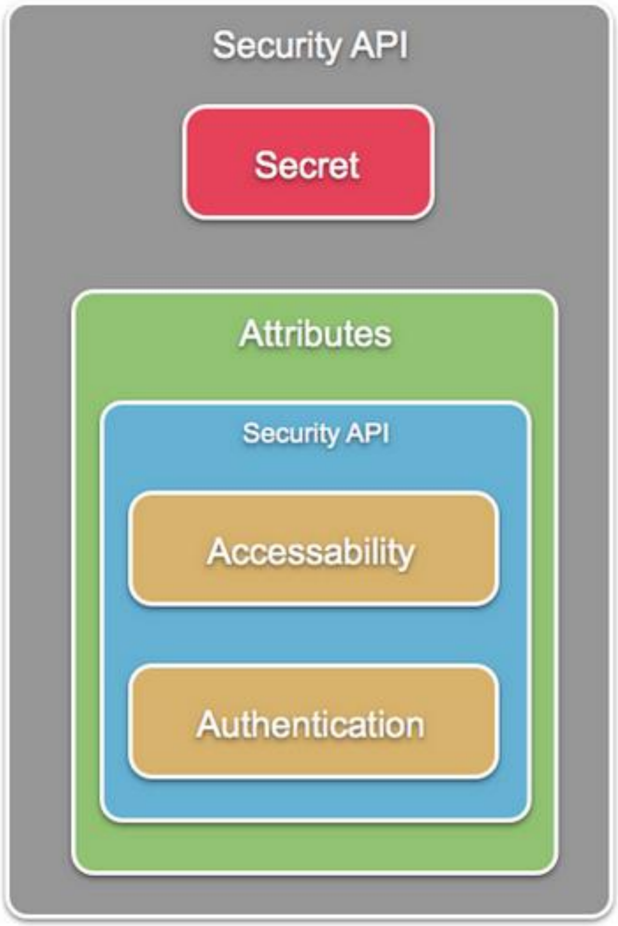
Uninstalling an app does not remove KeyChain data!

iOS KeyChain Items

Every entry has…

- Access control list (ACL)
  - **Accessibility**: When is item readable? → Policy
    Similar to protection class
  - **Authentication**: What authentication is needed for access?
- Key wrapped with protection class key,
- Protection class affiliation
- Attributes describing the entry See: https://goo.gl/iHYHOX
- Version number

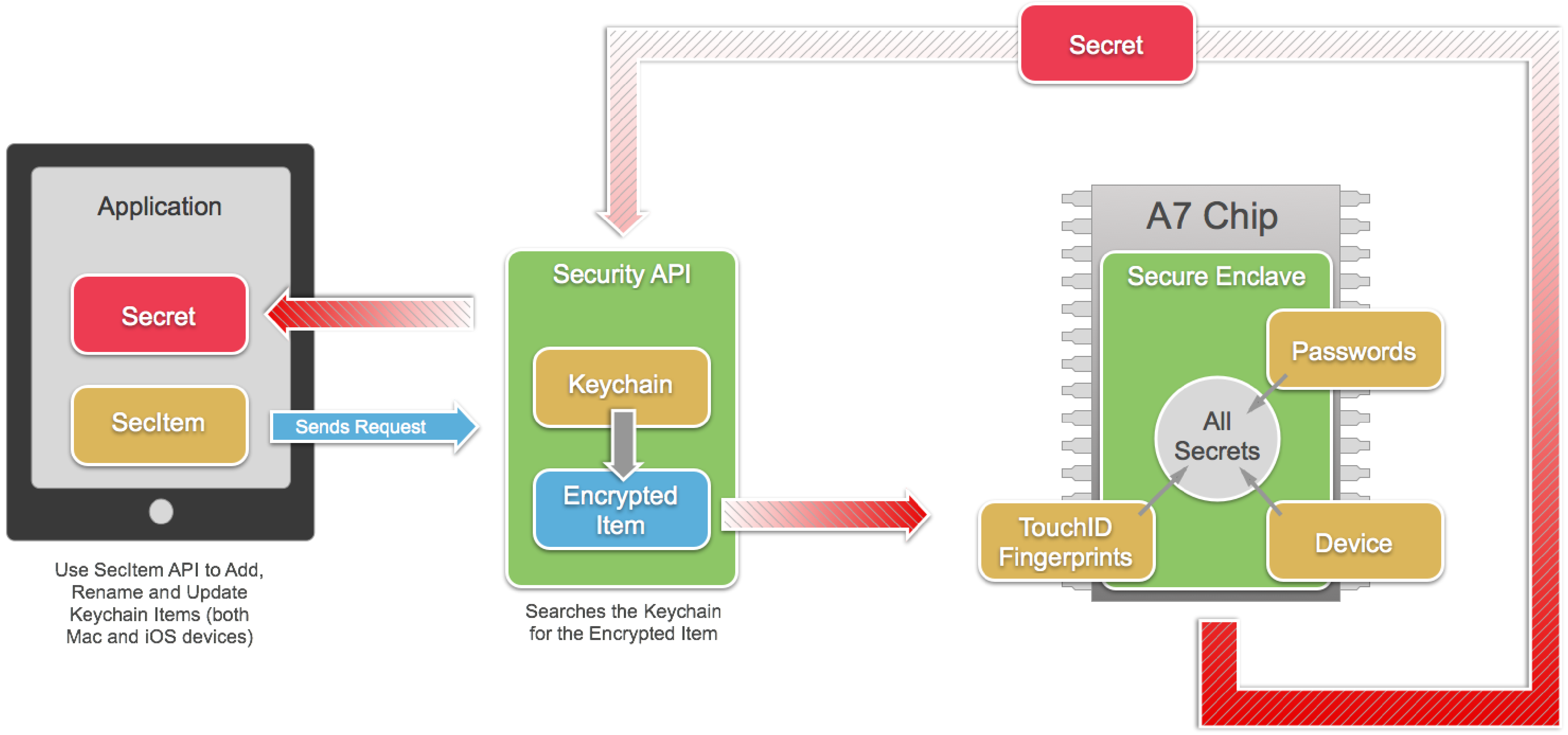
→ Every aspect is encrypted (AES-128 GCM)!
E.g. also usernames (= attribute), not only passwords!



Source: Xamarin

IAIK TU Graz

iOS KeyChain: App Access Workflow



Use SecItem API to Add, Rename and Update Keychain Items (both Mac and iOS devices)

Searches the Keychain for the Encrypted Item

Gets decrypted in the Secure Enclave

Source: Xamarin

KeyChain Protection Classes

Represent the policies to determine when keys are accessible

Key Availability	Key Data Protection
When unlocked	`kSecAttrAccessibleWhenUnlocked`
While locked	`N/A`
After first unlock	`kSecAttrAccessibleAfterFirstUnlock`
Always	`kSecAttrAccessibleAlways`
Passcode-enabled	`kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly`

Key Protection Classes

- `kSecAttrAccessibleWhenUnlocked`
  - Class key wiped when device is locked
- `kSecAttrAccessibleAfterFirstUnlock`
  - Class key remains in memory after first unlock
- `kSecAttrAccessibleAlways`
  - Class key remain in memory
- `kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly`
  - Class key wiped when device is locked but only if passcode is set
  - Never stored to backups or escrow keybags

Passcodes

Why is it useful?

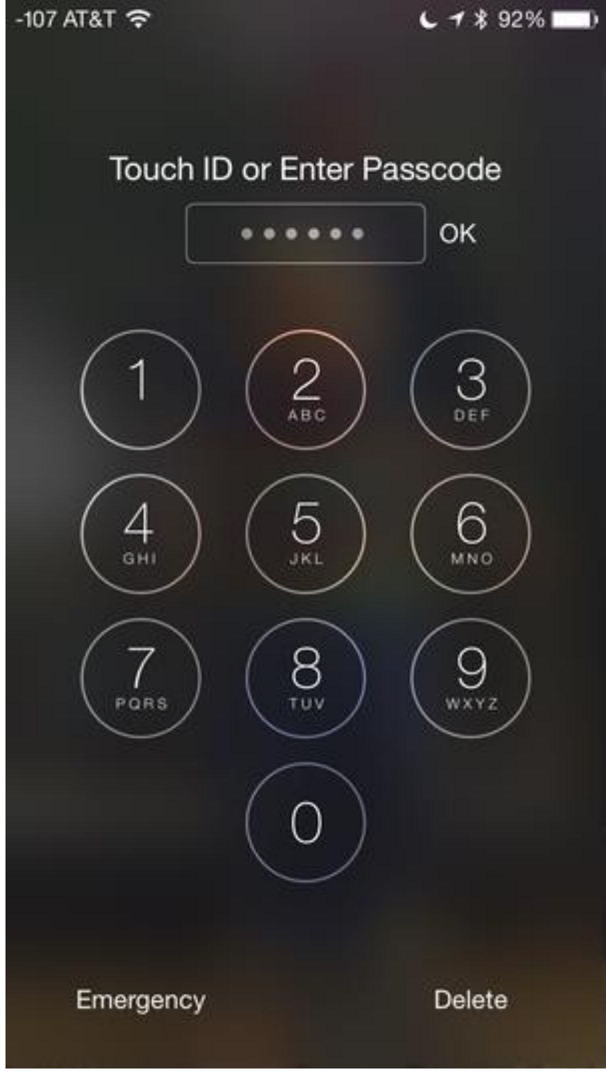
Enables file data protection!

What happens inside?

- User unlocks device with passcode
- Key is derived using PBKDF2
- Used to unwrap (decrypt) class keys

Can we attack it? :-)

Yes, but iteration count for KDF calibrated with device
Goal: One derivation should take ~80ms (newer: 100-150ms)



4 digits, 6 digits, arbitrary-length alphanumeric password

IAIK TU Graz

Passcode Attacking

Apple A7/A8 processor introduced delay

- KDF performed in Secure Enclave
- 5 second-delay added between unlock attempts!

Until iOS 9

- 4 digit pincode could be brute-forced in 14 hours
- Unlimited PIN attempts due to bug (CVE-2014-4451) See: https://goo.gl/lnKd3M

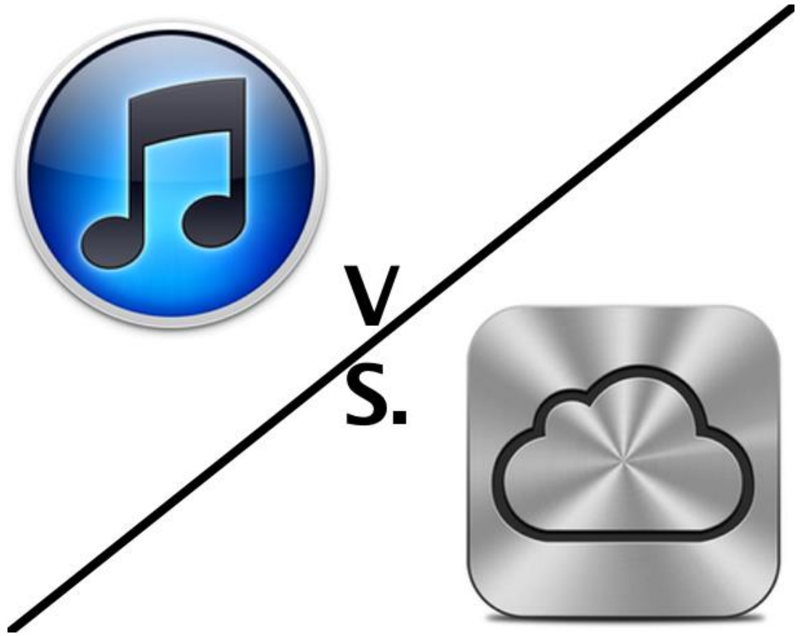
Now

- Additional delay introduced, e.g. 5 tries → 1min delay, 6 tries → 5min, ...
- If user-enabled, system wipe after 10 incorrect tries

Backups

Local or Remote Backup

- Local iTunes backups (WiFi or USB cable)
  - Encrypted (AES-256 CBC)
  - Plain
- iCloud Service
  - Sync data to iCloud for distribution on your iOS devices



Keybags on iOS

Keys for file and KeyChain Protection classes are managed in „Keybags“

- System Keybag
  - Contains all wrapped keys for protection classes
  - Ex.: Passcode entered →key for `NSFileProtectionComplete` is loaded

- Backup Keybag
  - Transfered (export) system keybag in backups
  - Backup <u>encrypted</u>: Key derived from iTunes password (10 Mio. iterations PBKDF2)
  - Backup <u>plain</u>: KeyChain still protected by UID-derived key
    → To migrate backup to new device: encrypt the backup!

Keybags on iOS

- Escrow Keybag
  - Used for iTunes syncing and MDM
  - Allows iTunes to backup and sync without user passcode!
    - Each device generates iCloud KeyChain synchronization key pair
    - User explicitly approves new devices joining the „sync circle“
  - Needed for every OTA update (user is prompted for passcode)
  - Together with accepting provisioning profiles
    → weakest point in OS!

iCloud Backup

- Encrypted using backup („escrow“) key
  - Randomly generated key
  - Wrapped using a key that is derived using a KDF from an „iCloud Security Code“ (iCSC)
    → iCSC = User passcode

- Backup encrypted with escrow key
  - Sent to Apple in wrapped form
  - In case of device loss or new device
    → User can recover secrets with iCloud password and iCSC

- Main problem in practice: iCloud account security

iCloud Backup

„Who watches the watchers?“ :-)

Other weaknesses?

iCloud backend could brute-force iCSC to access escrow key!

Apple‘s solution: Cloud Key Vault

- Enforce policy over escrow key
  - Want hard limit on escrow recovery attempts under adversarial cloud
  - What if escrow key unwrapping only happens in Hardware Security Modules?
- Cloud Key Vault = HSM running custom secure code
  - Key vault runs own certificate authority
    - Private key never leaves HSM
  - Each iOS device hardcodes key vault CA cert