IAIK TU Graz

Key & Data Storage on Mobile Devices

Mobile Security 2022

Florian Draschbacher
florian.draschbacher@iaik.tugraz.at

Slides based on those by **Johannes Feichtner**

Outline

- Why is this topic so delicate?
- Keys & Key Management
- High-Level Cryptography
- Practical examples for key handling

Pictures: Google / Apache 2.0

„Where do I store my key?“

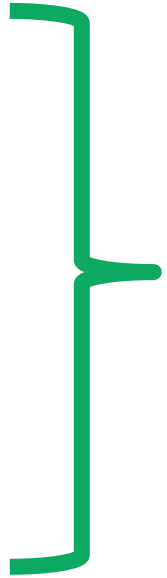
„Where should an application store a key?“

About Keys

- Sensitive information by design
  - Kerckhoff‘s principle: „*only secrecy of the key provides security*“
    → Need for secrecy fundamental!
  - „Trust“: Revocation, Certificate Chains, PKI

- Sharing, exchanging keys
  - Key Agreement, protocols, multiple devices

- Storing, using, replacing keys
  - Applications (Password Managers, Secure Messengers, …)
  - Storage location (local, remote → cloud?)



Key Management!

Key Management

Generating, using, storing, exchanging, replacing keys

- Historically grown pain - often due to proprietary solutions!
  - Meets enterprise requirements?
    - Backup, Key destruction, monitoring usage
  - Compliance with policies, e.g. PCI DSS
- Classical solution: Public Key Infrastructure (PKI)
- Nowadays: No longer „all or nothing“ approach
  - (Network-)Interoperability protocols, e.g. OASIS KMIP
  - Increasing standardization, e.g. JSON Web Keys (RFC 7517)

Key Management Today

- Increasing customer demand for data confidentiality
  - Snowden revelations made people think
  - Data security appears in marketing, e.g. Apple:

> “Apple has never worked with any government agency from any country to create a backdoor in any of our products or services. We have also never allowed any government access to our servers. And we never will.”

Source: apple.com

… but smartphones were never built as data safes!

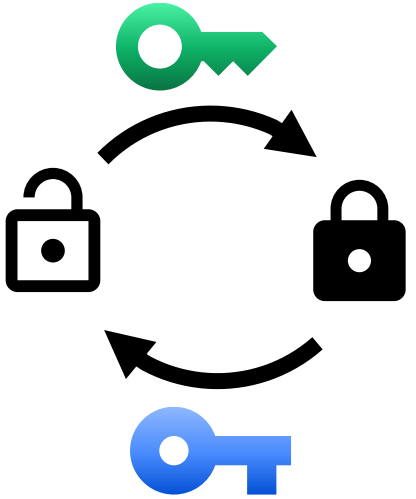
→ Can we get a „high level of security“ on „cheap“ Android / iOS phones anyway?

Note: Key Management is more than encryption!
Digital signatures, identity verification, authentication, …

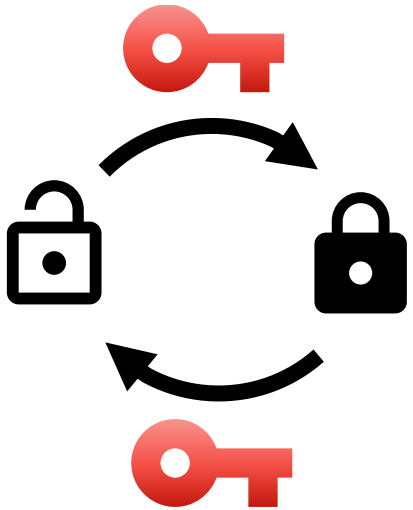
Our focus: Key storage & Data encryption

Basics - Asymmetric vs. Symmetric

Before even thinking about storing the key…



- Asymmetric cryptography
  - Slow, cannot be used for bulk data encryption
    e.g. for encrypting files, data, messages, etc.
  - Used for wrapping symmetric keys



- Symmetric cryptography
  - Fast, used for bulk data encryption
  - Key exchange?

Key Storage on Smartphones

Problem

- Encrypting files on the platform / device
- In theory
  - Use some symmetric algorithm, e.g. AES
  - Encrypt / decrypt files
  - Store the key in a secure way

} How to store the key securely?
How to exchange key material?

Ingredients

- **Data:** Payload to encrypt / decrypted
- **Encryption engine:** Handling actual transformation process
- **Key manager:** Handling keys, passing to encryption engine

Basics – Hybrid Cryptosystems

Basic idea

Combine convenience of public-key system with efficiency of symm. schemes

→ **Outcome:**

- Fast cryptosystem, suited for big data
- No key exchange needed before sending, only public key of recipient

And in practice?

- TLS
- OpenPGP
- PKCS#7: Cryptographic Message Syntax (CMS)

Basics – High-Level Crypto

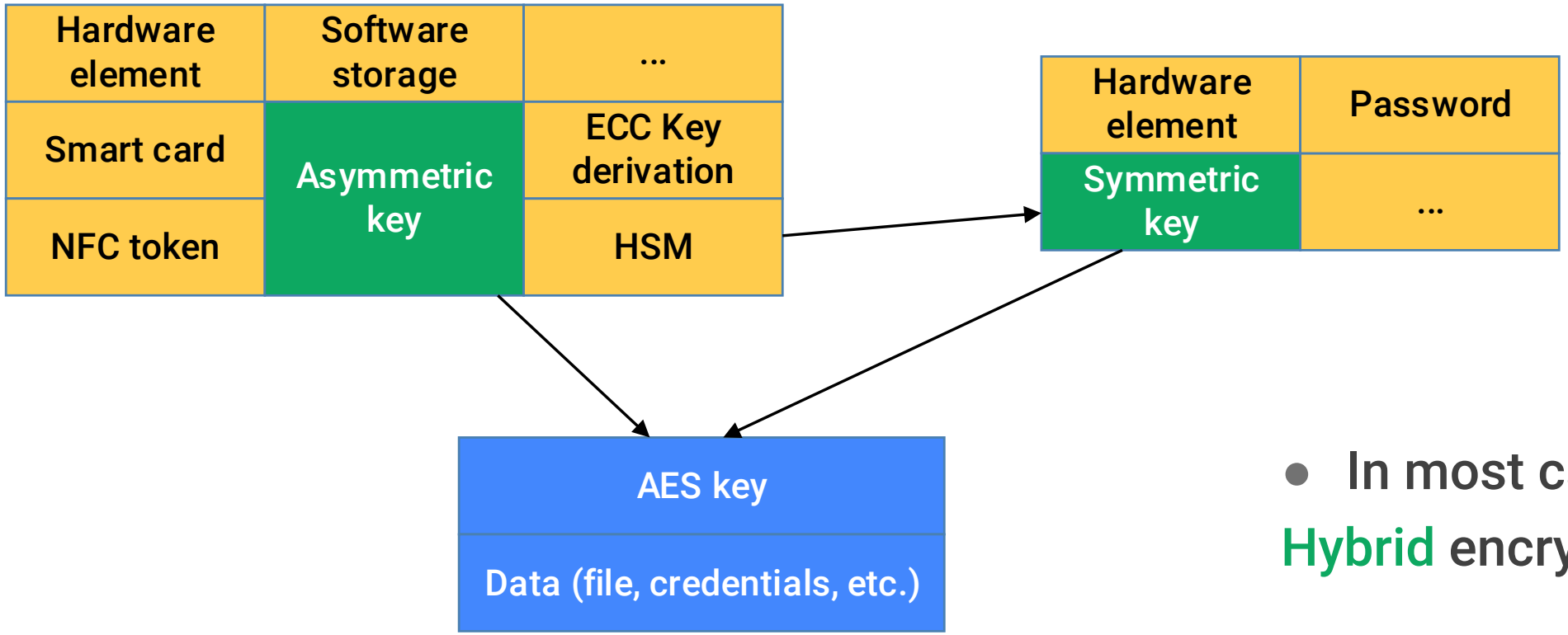
Cryptographic Message Syntax (CMS)

- Used by S/MIME
- Encryption
  - Defines format for hybrid data encryption scheme
    ASN1, multiple recipients, many algorithms, modes, …
- Signatures
  - Format for storing hash values, many algorithms, etc.

XMLEnc, XMLDSig

- Signing / Encrypting XML documents or specific elements
- Used for Austrian Citizen Card / Mobile Phone Signature

Key Storage on Smartphones



- In most cases:

Hybrid encryption schemes

- Local storage also:

Symmetric keys only

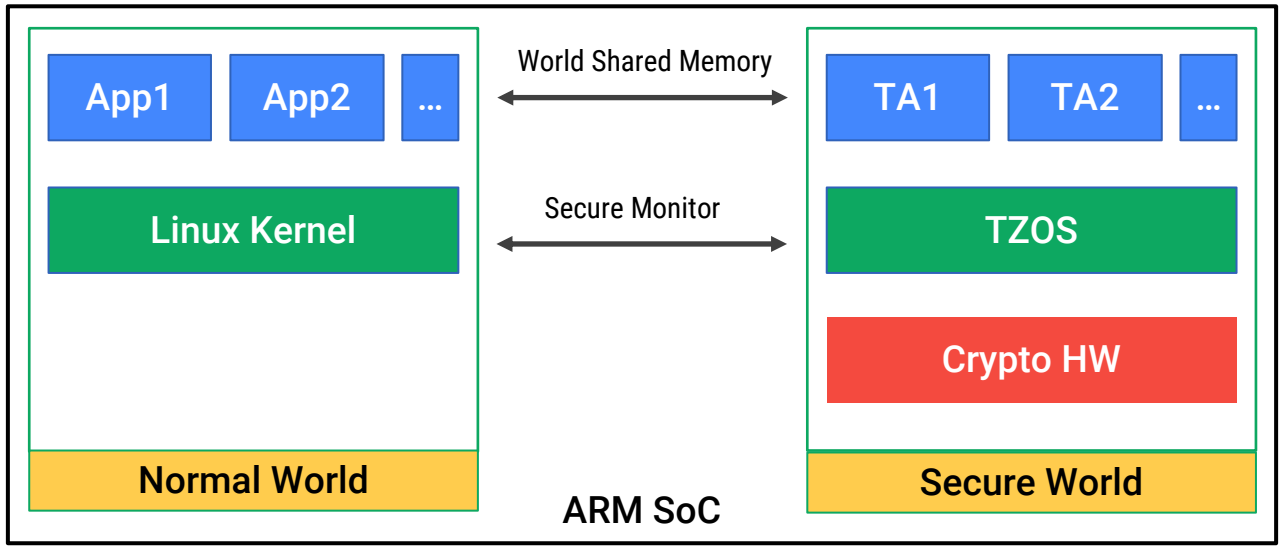
Practical Examples

ARM TrustZone

What is it?

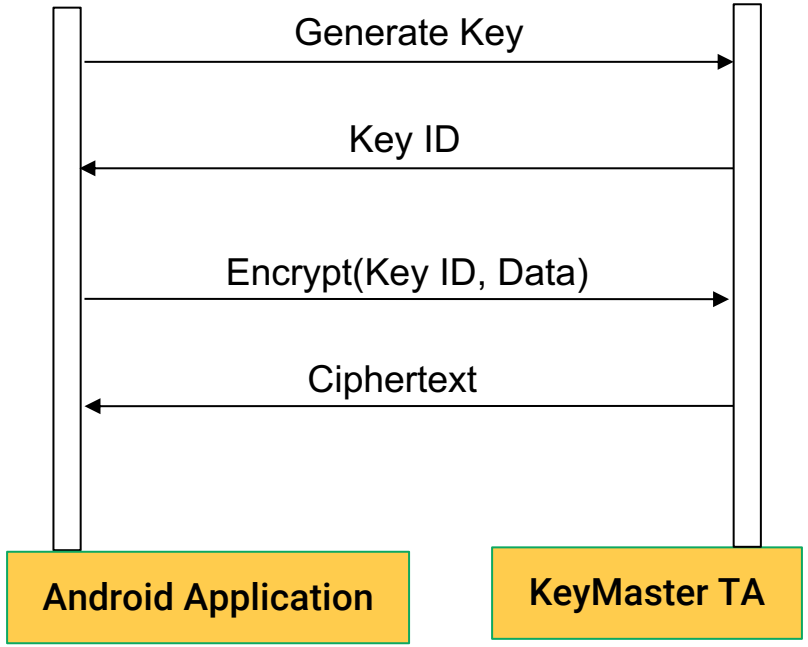
- A Trusted Execution Environment (TEE) inside modern ARM CPUs
- “Secure World” almost completely isolated from “Normal World”
- Executes some TrustZone OS and Trusted Applications
- One of them: KeyMaster TA (Android)
  - HW KeyStore implementation

- Can access HW crypto engine



KeyMaster Trusted Application

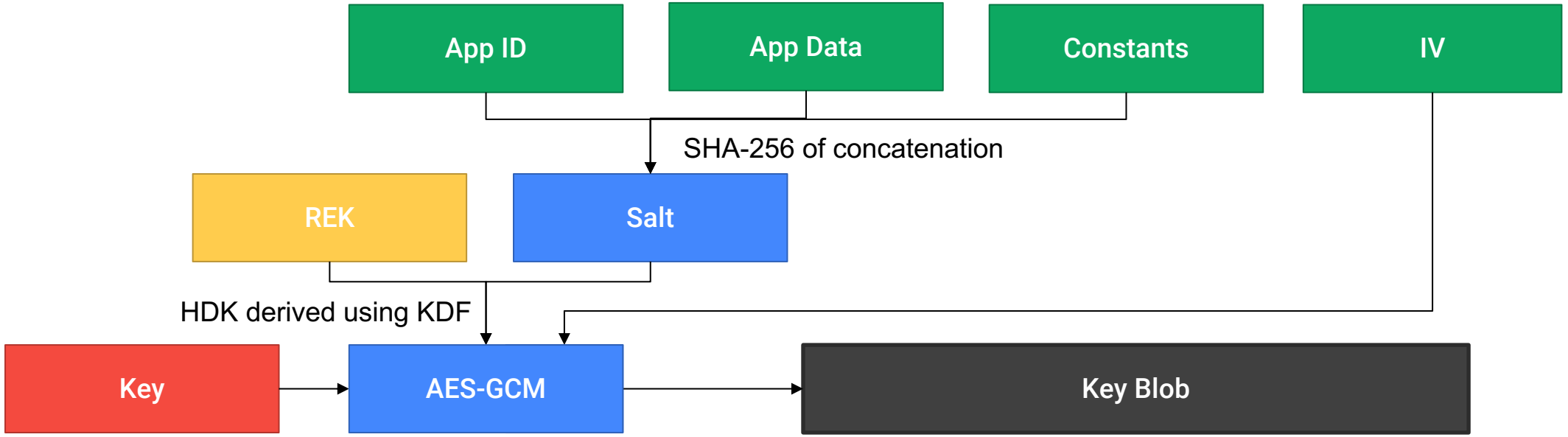
- Implements a HW-backed Android KeyStore provider
- Generating, importing and storing cryptographic keys
- Encrypting, Decrypting, Signing, Verifying, …
  - Keys never leave the TrustZone
  - Secure even if Android OS / kernel compromised
- Request user authorization
  - Fingerprint, …



“Keys never leave the TrustZone”

Well, actually…

- Keys are wrapped and stored in Normal World
  - ”Key Blobs” encrypted by Hardware Derived Key (HDK)
  - HDK derived from permanent device-unique Root Encryption Key (REK)



Flawed key wrapping

- KeyMaster Interface is defined by Android OS
  - KeyMaster HAL (Hardware Abstraction Layer)
- The implementation is not !
  - Multiple different implementations
  - Even for a single vendor
- 2022: Serious issue in Samsung implementation
  - Allows key extraction on rooted devices

How could that happen?

Forbes Subscribe Sign In

Feb 24, 2022, 05:58am EST | 6,130 views

Serious Security Shock For 100 Million Samsung Galaxy S8-S21 Smartphone Users

Davey Winder
Senior Contributor
Cybersecurity
Co-founder, Straight Talking Cyber
Contrib. Editor, PC Pro Magazine

Follow

Listen to article 7 minutes

If you have a Samsung Galaxy flagship smartphone from the S8 onwards, I have a serious security shock to impart: hackers found a way of extracting security keys and the most highly sensitive data protected by them from these flagship devices. In all, 100 million Samsung smartphones across five generations were impacted by a double-whammy of high-severity vulnerabilities determined to be exploitable by a team of security researchers.

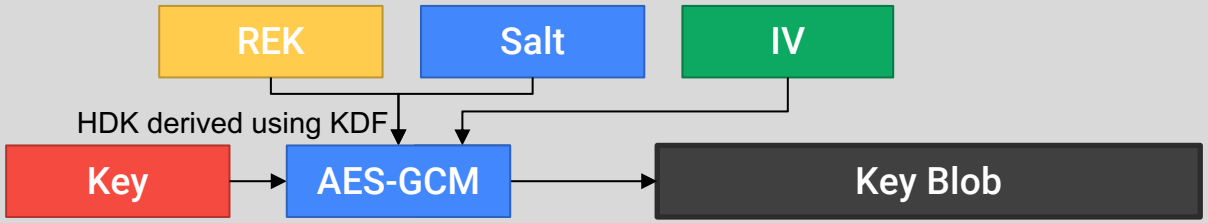
Source: forbes.com

Samsung KeyMaster Flaw (CVE-2021-25444) (Source: Shakevsky et al., 2022)

- Samsung implementation allows IV to be controlled from “Normal World”
  - As well as all values used for deriving the salt

- Given key blob B_A of unknown key K_A, Import new blob B_B wrapping known K_B
  - *E(HDK, IV):* Key-Stream created from key HDK and init-vector IV

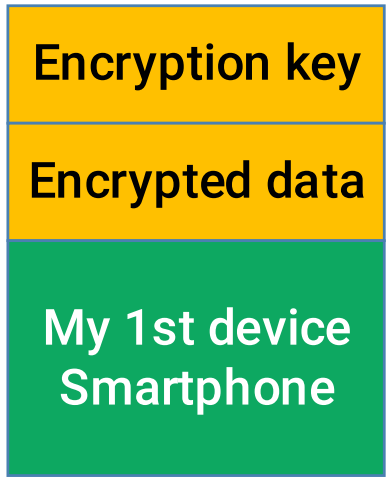
$$B_A \oplus B_B \oplus K_B = (E(HDK,IV) \oplus K_A) \oplus (E(HDK,IV) \oplus K_B) \oplus K_B =$$
$$K_A \oplus K_B \oplus K_B = K_A$$



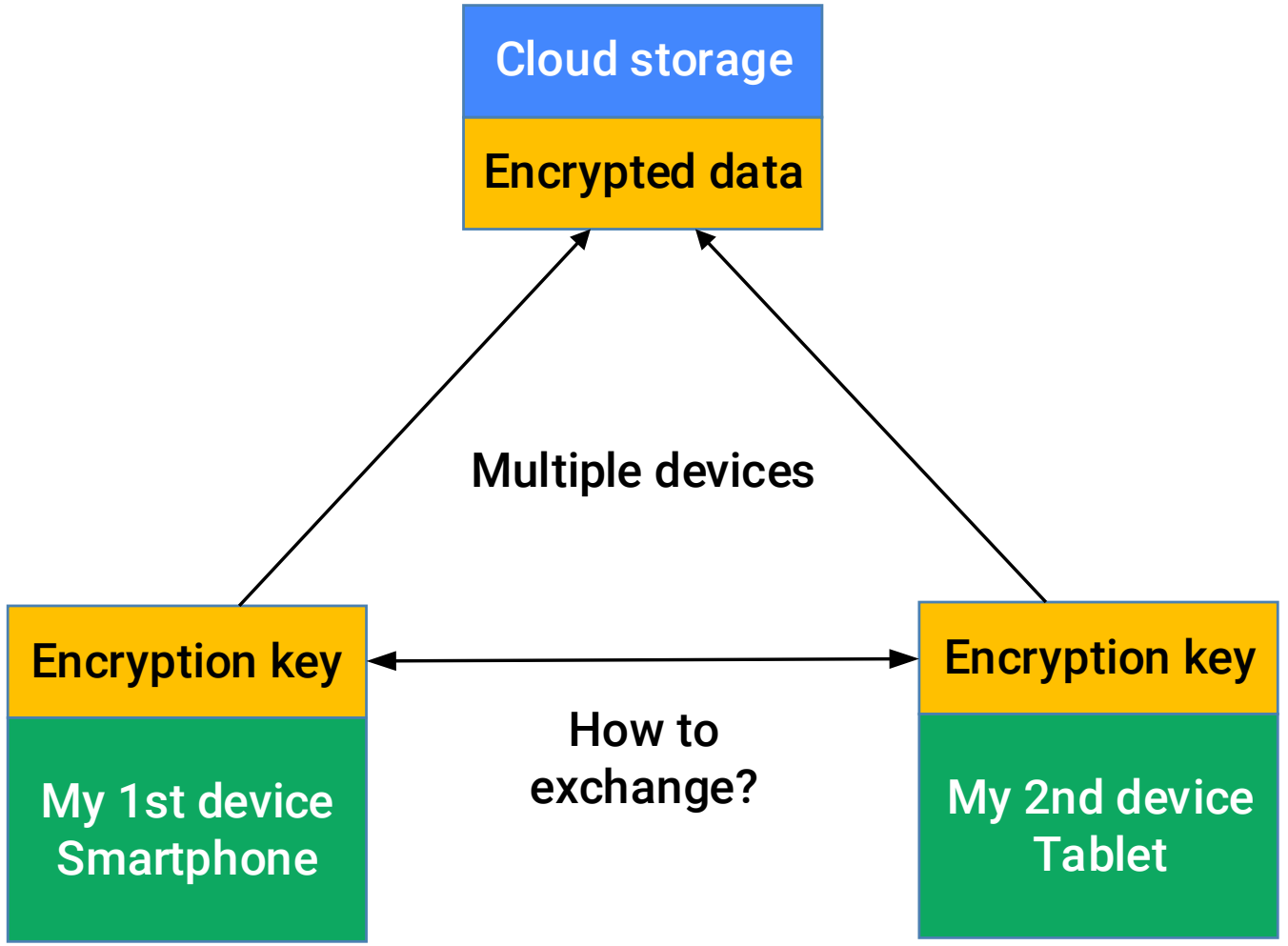
Key Management Scenarios

- Scenario A: „I want to keep my secret data locally on my phone“
- Scenario B: „I want to share encrypted data over multiple of my devices“
  Cloud-based password manager, encrypted cloud storage, …
- Scenario C: „I want to exchange encrypted data between different users“
  Via CMS / SMIME, a secure messaging app, …

Scenarios A & B

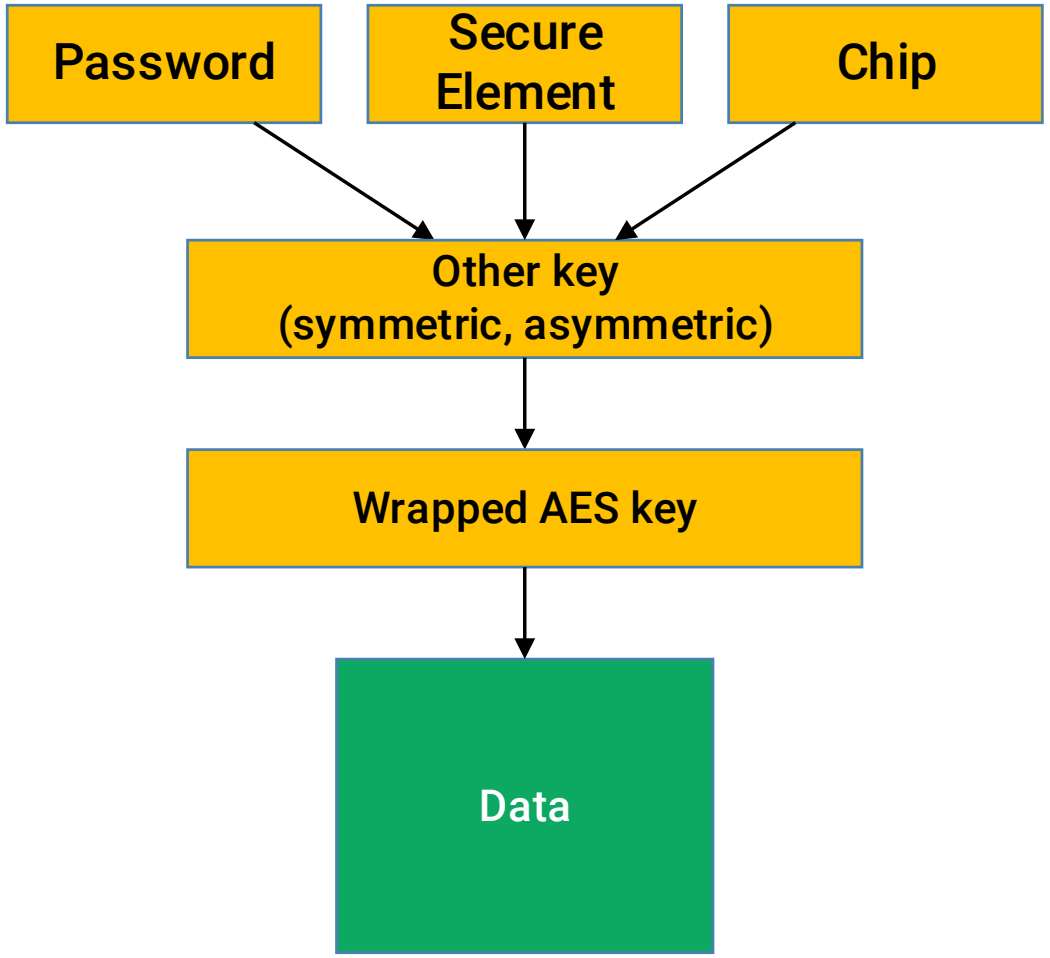


Scenario A



Scenario B

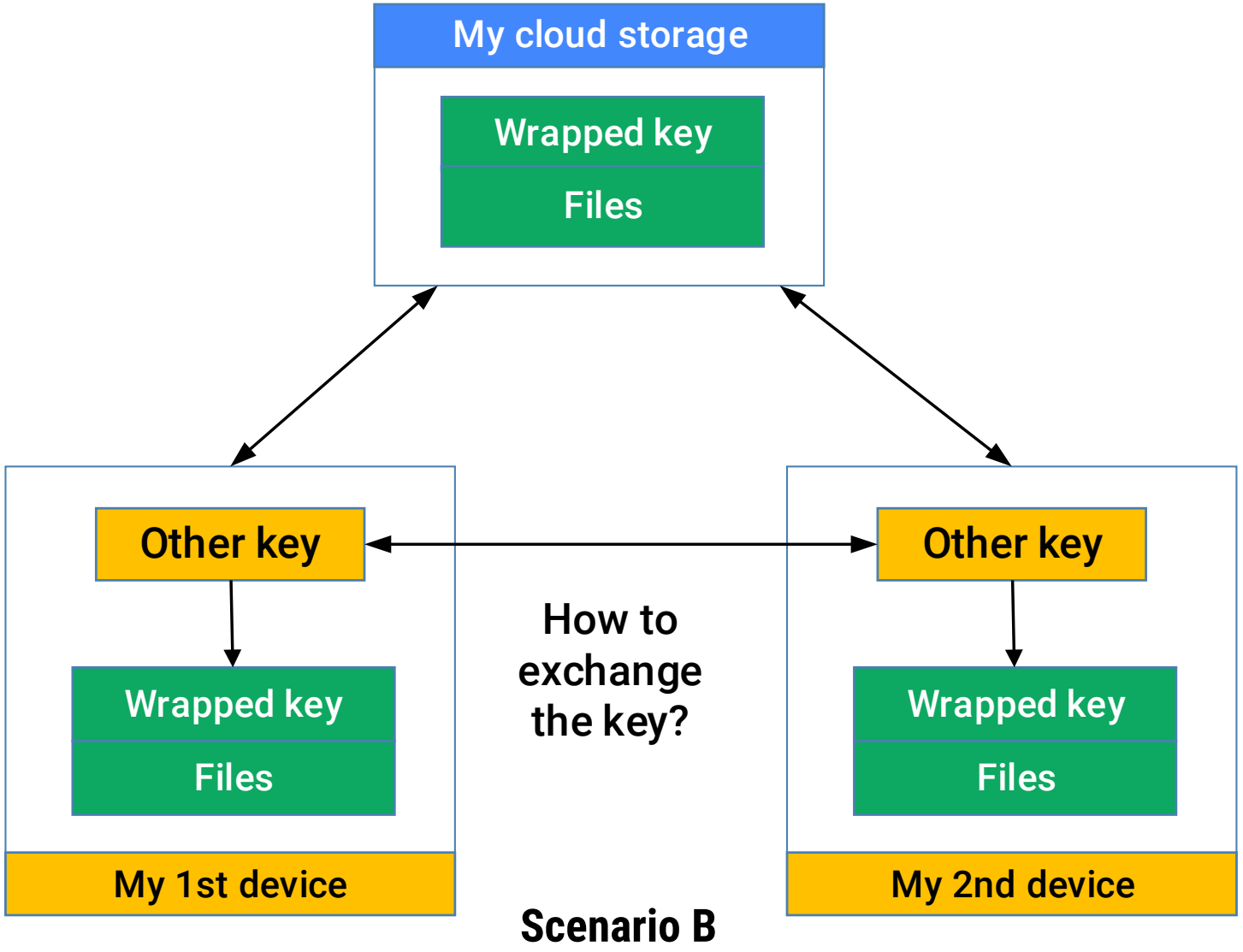
Scenario A: Keeping data locally encrypted



Scenario A

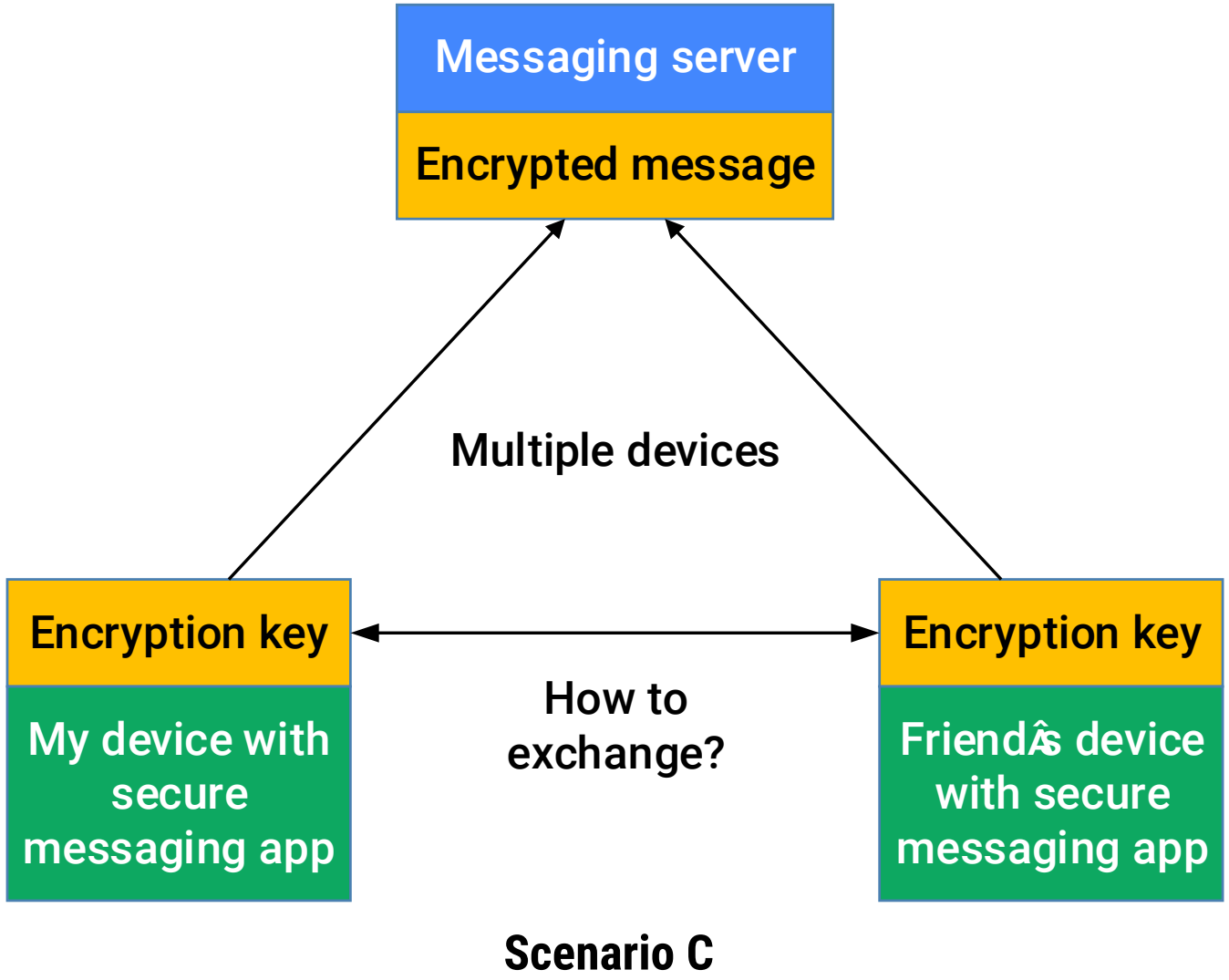
- Typically there is *key hierarchy*
- Actual data encryption key wrapped with another key
- Much easier to change key when GBs of data are encrypted
- Wrapped key stored together with file (key is encrypted)

Scenario B: Share encrypted data (1 user)



- Wrapped key stored together with files on storage provider
- But how to exchange the key in a secure way?
- Since you own both devices → export, import manually quite tedious…

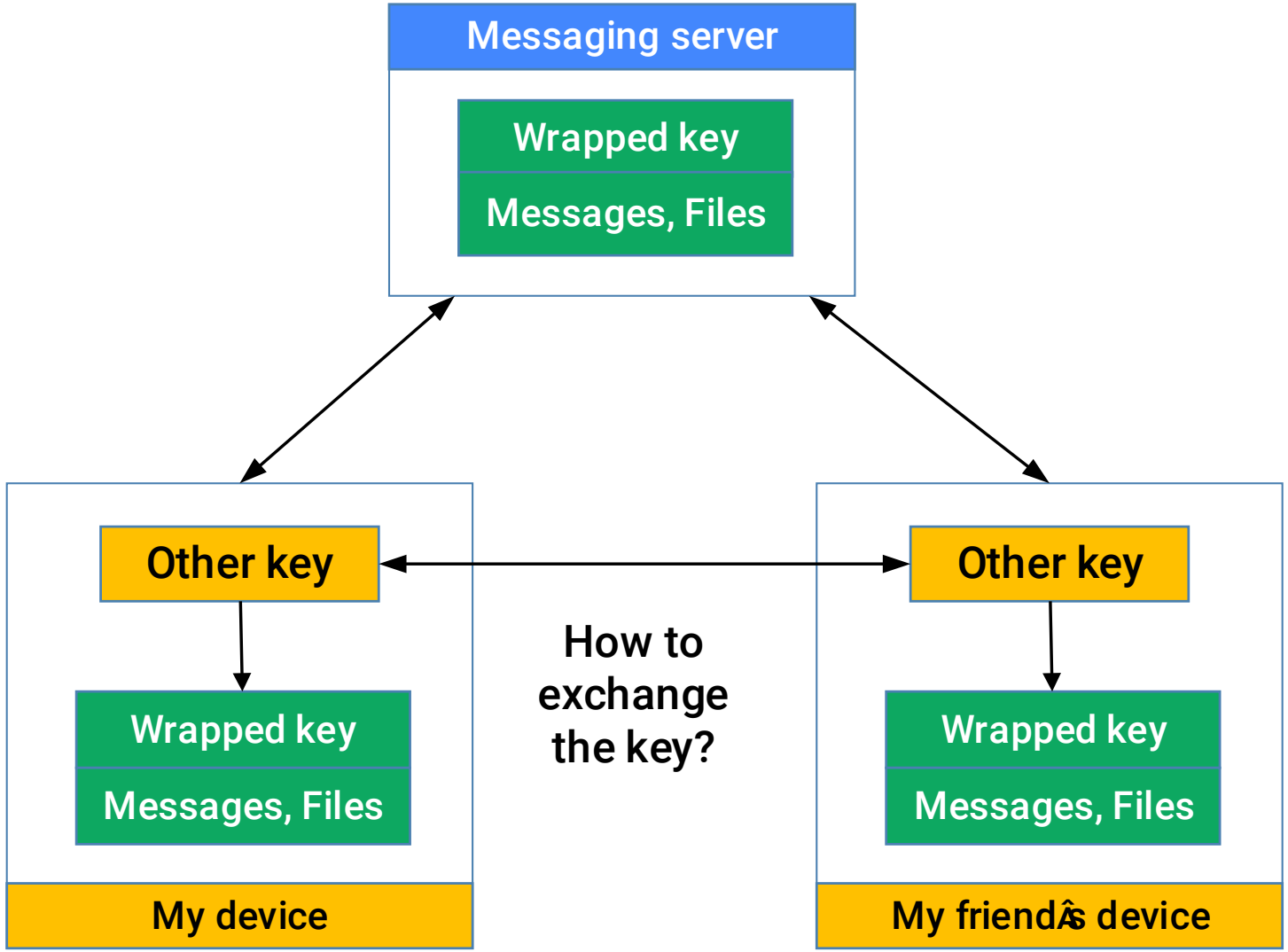
Scenario C



Scenario C

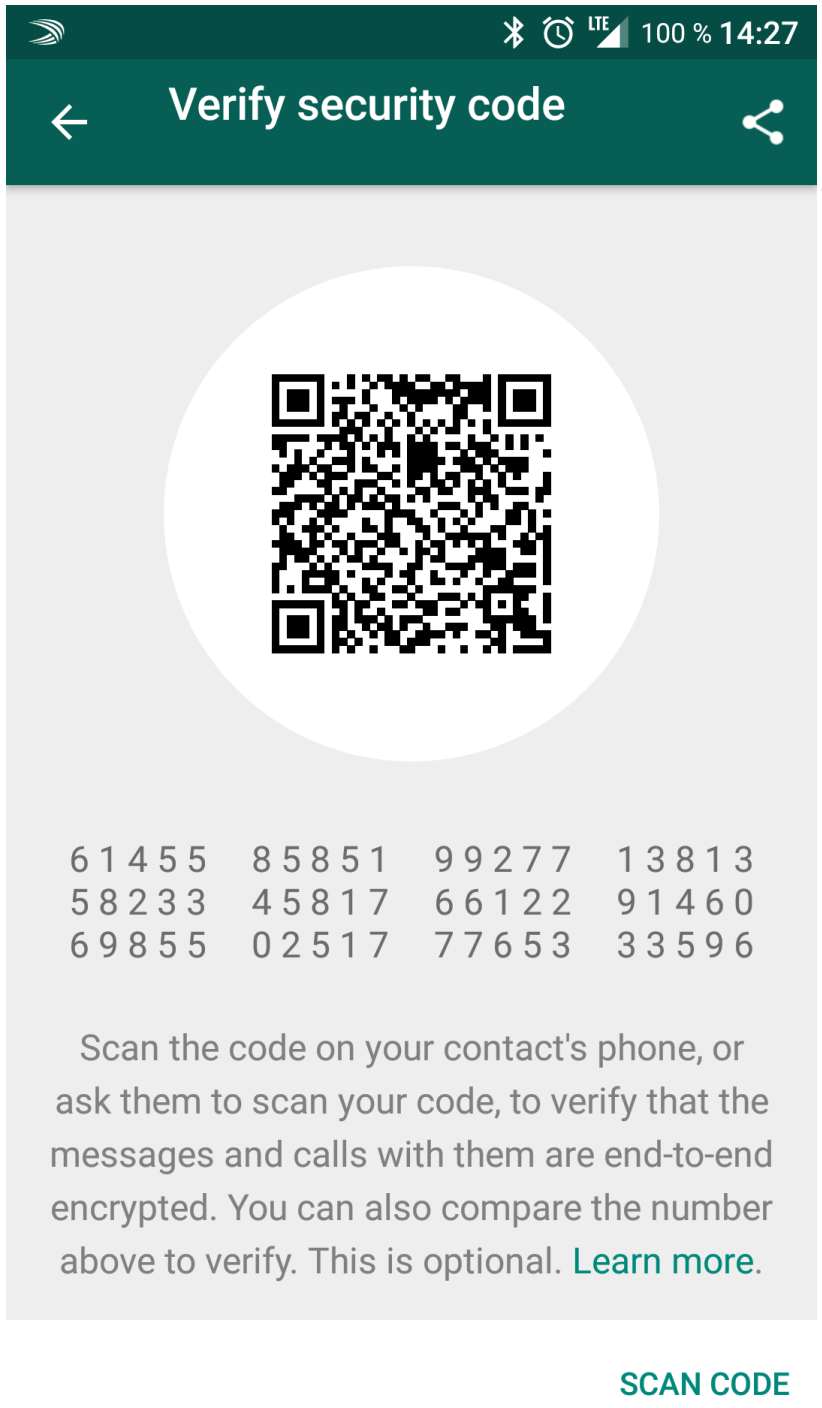
- Apple iMessage
- BlackBerry Messenger
- CMS / SMIME
- Secure messaging apps
  - Signal (WhatsApp)
  - Threema
  - Telegram
  - ...

Scenario C: Share encrypted data (multiple users)

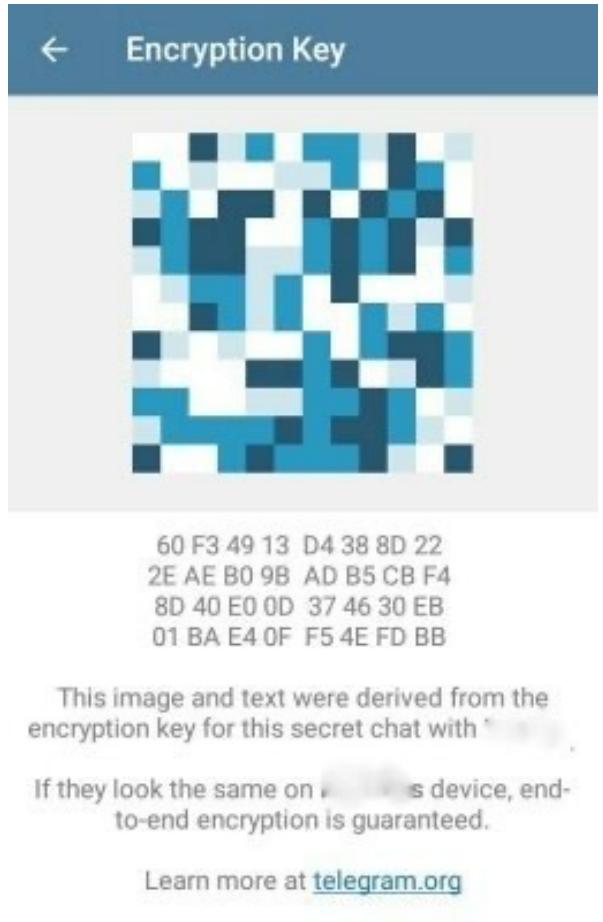


Scenario C

- Basically same problem as in Scenario B
- However, key exchange with another person
- How to trust this person?
  - Out of scope for today
  - Many things to consider!



Scenario C: Share encrypted data (multiple users)



Approaches

- ***Manually:*** Verify public key
  - WhatsApp, Telegram, …
- (Indirect) 3rd party trust
  - Trust person because you trust group / company admin
- ***Automated:*** Key servers
  - PGP, S/MIME, Keybase, (WhatsApp)…
  - Open question: „Really the key of person X?“

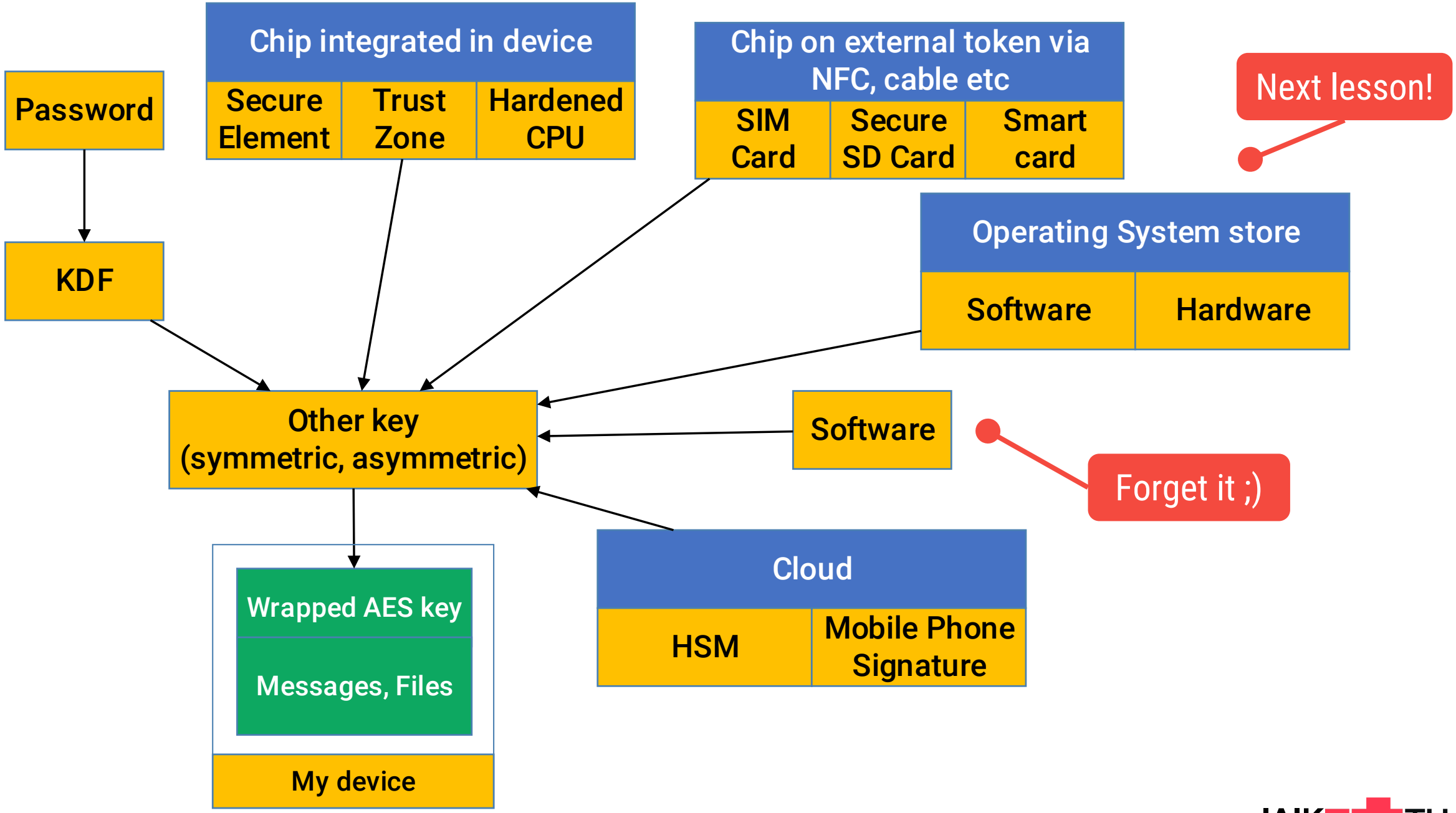
IAIK TU Graz

Focus on Scenario A & B

How to „wrap“ the actual file encryption key?

- Directly, with another key
  - Asymmetric: RSA, ECDSA
    - Especially relevant for Scenario C, but also used for A & B
  - Symmetric: AES, ECIES
    - For Scenario A & B ok since we do not require to exchange key with another person
- Indirectly, with another key, derived from a passcode
  - Heavy usage due to simplicity
  - Elimininates the key exchange problem

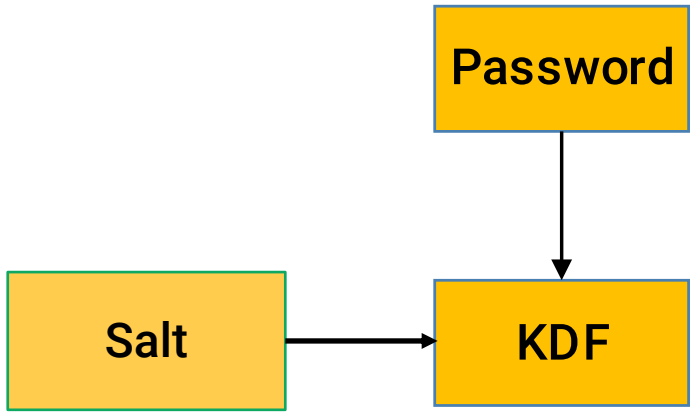
Key Storage Zoo


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
Next lesson!
Password
KDF
Operating System store
Software
Hardware
Other key (symmetric, asymmetric)
Software
Forget it ;)
Cloud
HSM
Mobile Phone Signature
Wrapped AES key
Messages, Files
My device

Passwords

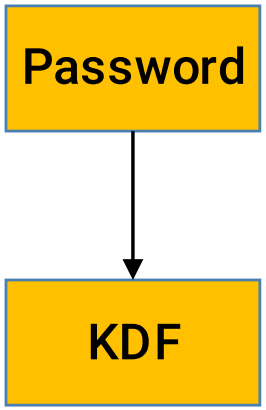
Never directly use a password as a cryptographic key!

- Human-readable
  - Much too low entropy
- Solution: Key Derivation Functions
- Use salt for preventing dictionary attacks



Passwords

Secret undergoes key derivation, e.g. 20.000 HMAC iterations
→ Result: Cryptographic key



\+

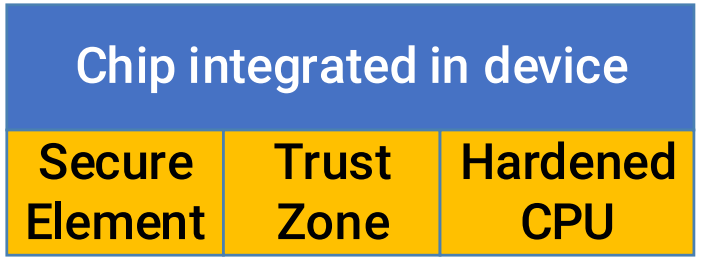
Very simple method, no key exchange required,
no complex hardware token etc

\-

- Many things can go wrong:
  Key derivation function, Brute-force attacks
- Usability vs. Security! E.g. short passcodes on mobile devices

```
new PBEKeySpec(password, salt, iterationCount, keyLength);
```

Integrated Chips



- Special chips integrated on mobile devices, PCs, etc.
  Used to store store cryptographic keys (symmetric, asymmetric)

- Dedicated crypto hardware protected against attacks because
  by design key cannot be exported / extracted

- Very good solution for device encryption systems, e.g. Android / Apple Pay

But: How to deploy keys? How to exchange them? Use over multiple devices?

Summary: No data protection for multiple devices, e.g. cloud storage

Integrated Chips

Chip integrated in device		
Secure Element	Trust Zone	Hardened CPU

+

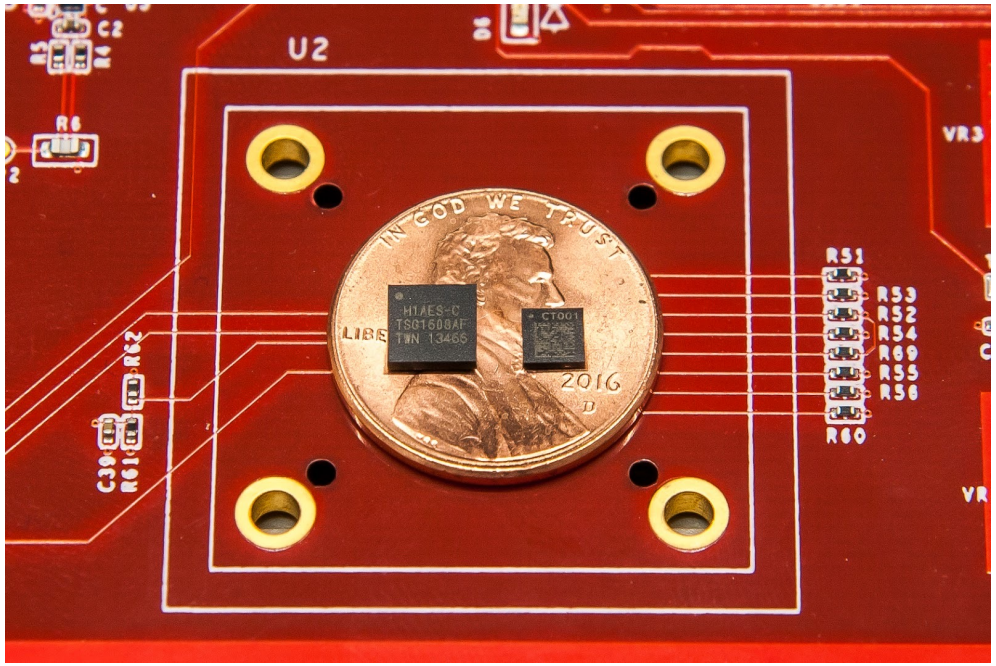
- Used to create very secure file encryption systems on mobile devices
- Very good use for specific problems, e.g. Trusted Boot, Android Pay

-

- How to use over multiple devices?
- How to exchange / deploy keys? *(no access to integrated SEs!)*
- Key distribution!

Secure Element

Case Study: *Google Titan M*



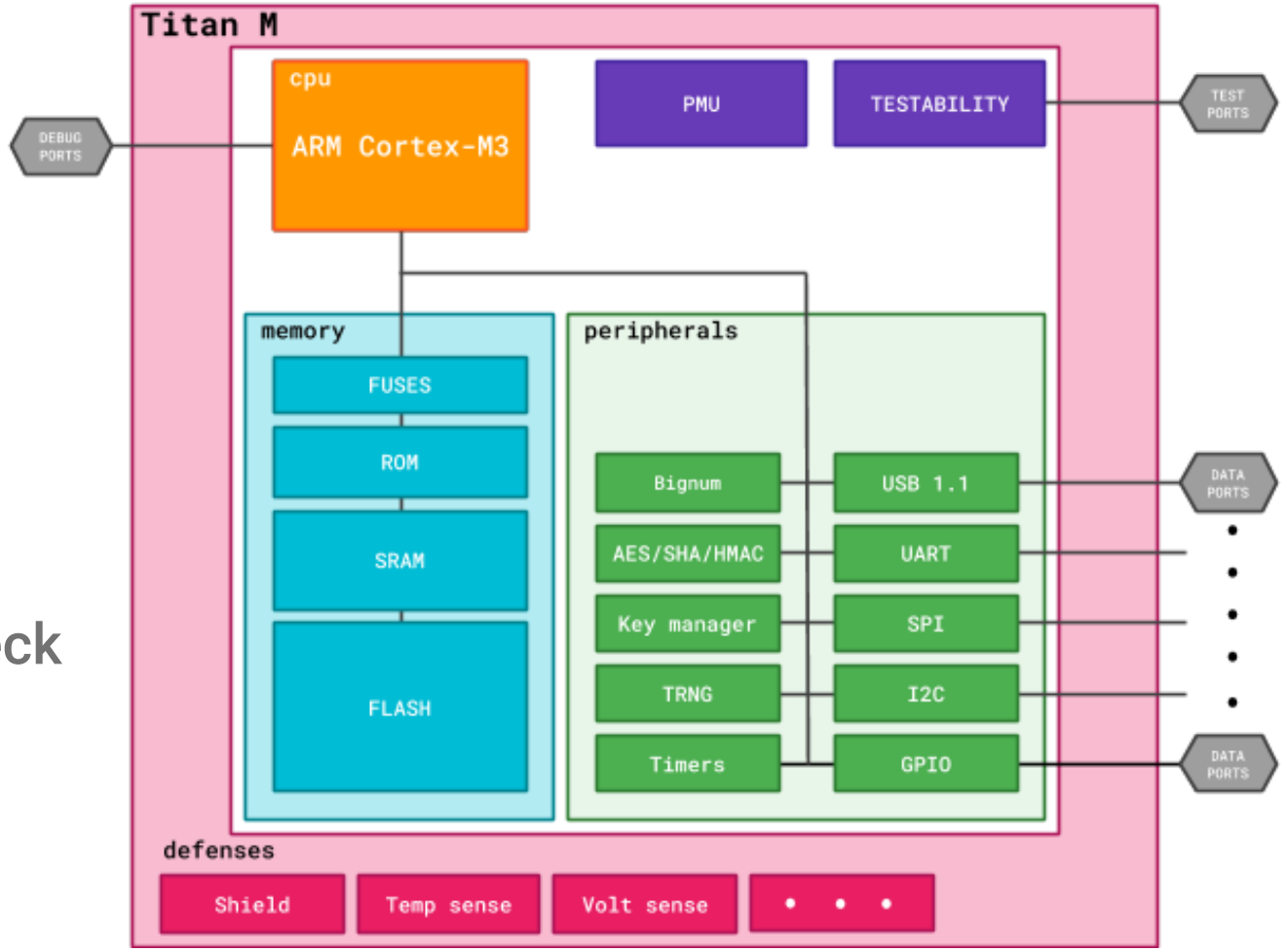
- *„From silicon to boot“*
  - Own supply chain & manufacturing process
  - Own design of logic gates to boot code

- Open-source firmware
  - Only signable by Google
  - Verifiable binary builds
  - „Insider Attack Resistance“ should prevent forced firmware updates →key extraction

Secure Element

Case Study: *Google Titan M*

- ARM Cortex-M3 CPU
  - 64 KB RAM
  - Flash read-only after signature check
- HW accelerators
  - AES / SHA / HMAC
  - Big number coprocessor for public key algorithms
  - Initial key provisioning using entropy by True Random Number Generator (TRNG)



Source: googleblog.com / CC BY 2.5

External Tokens

Chip on external token via NFC, cable etc		
SIM Card	Secure SD Card	Smart card

Protected chips on external devices, contain cryptographic keys

- Smart cards
- SecureSD cards (SD slot!)
- SIM cards
  - Used by some signature solutions in Europe
  - Can be employed as kind of Secure Element
- Special tokens
  - Yubikey (FIDO U2F)
- NFC or cable

Picture: Bortsch / Public Domain

External Tokens

Chip on external token via NFC, cable etc		
SIM Card	Secure SD Card	Smart card

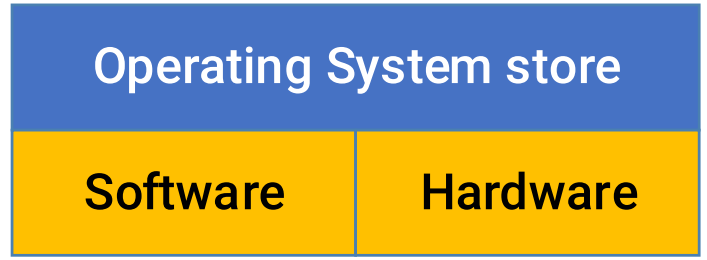
+

- Can be used on multiple devices
- In many cases: Own keys can be deployed
- Not too expensive (for dedicated solutions) and secure!
- Vital for payment systems, access control etc

-

- Usability! Drivers, support on different platforms (NFC on mobiles, Desktop, …?)
- How to deploy keys? Key distribution!
- Experience with Austrian Citizen card: Many problems with drivers, hardware, …
- Too expensive (e.g. supply card to every single citizen)

Operating System Store



We focus on mobile devices…

- KeyChains, file-encryption APIs to protect keys, passwords, files, etc
- Data storage in software or hardware
  - Software KeyStore still more secure than own app implementation
    → OS has deeper access to system, better protection a priori
- Accessible to developer

***But still:** Different solutions on different platforms!*

Software

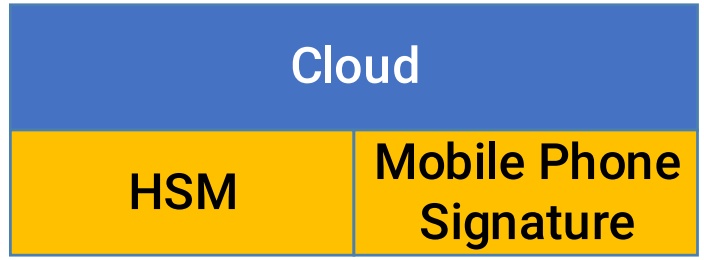
Software

- Storing the key material directly on the file system
  - Private folder of your mobile app
  - Either protect key via passcode (KDF) or
  - With transparent file encryption (iOS): Store it in plain
  - **In general: forget it…**

Note: Basically, we face the same situation in web browsers!

  - Only way to store web keys would be in HTML5 storage (W3C crypto API)
  - Or on an external token (FIDO U2F or in ancient times: Java Applets)

Cloud Storage



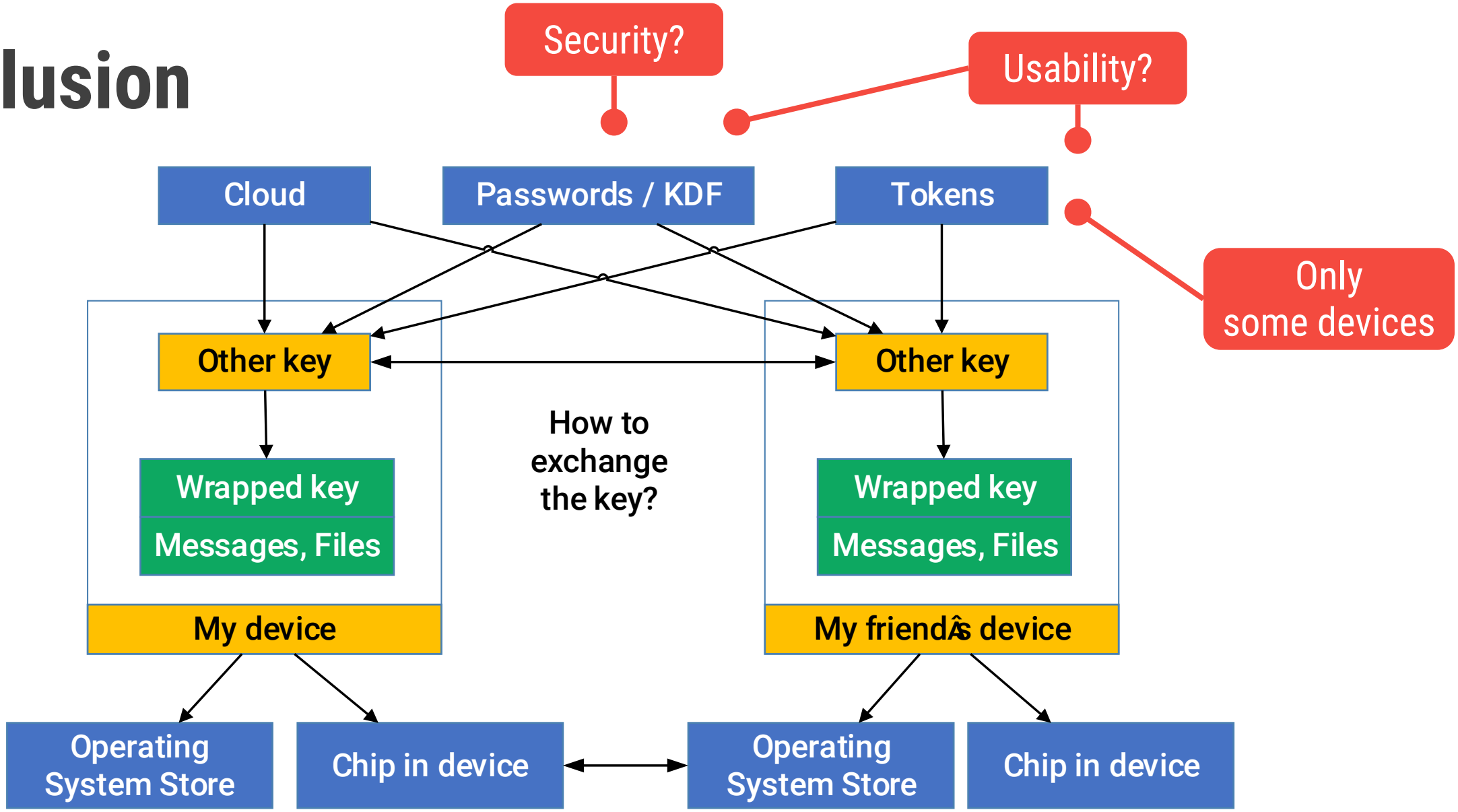
Basic idea: **The cloud as virtual smart card**

- Store keys in public or private cloud (protected by HSMs)
- Provide cryptographic functions over network with strong authentication
  e.g. Microsoft Azure KeyVault

See: https://goo.gl/g6tAAg

- Cloud solution
  - Amazon Cloud HSM, Cloudflare Keyless SSL
- Austrian Mobile Phone Signature

Conclusion


Security?
Usability?
Only some devices
Cloud
Passwords / KDF
Tokens
Other key
Other key
How to exchange the key?
Wrapped key
Messages, Files
Wrapped key
Messages, Files
My device
My friendâs device
Operating System Store
Chip in device
Operating System Store
Chip in device