Data Security

Mihai Ordean
Designing Secure Systems
University of Birmingham

How to prevent a rollback attack?

How to prevent a rollback attack?

- Counter based version control
- Blacklist based version control
- eFuses
- Apple nonce based protocol (i.e. APTicket): random unique value generated at every restore and signed by Apple
- …

Overview

- Device security
  - Is code on the device vulnerable to exploits ? (e.g. buffer overflows)
  - Is the code authenticated ? (i.e. has not been tampered with)
- **Data security**
  - Is the stored data is accessible to everyone? (e.g. encrypted)
  - Is the stored data authenticated?
- Metadata security
  - What does metadata reveal about data?
  - Can we tamper the metadata?
- Protocol security
  - Is data in transit visible?
  - Can data in transit be tampered with?

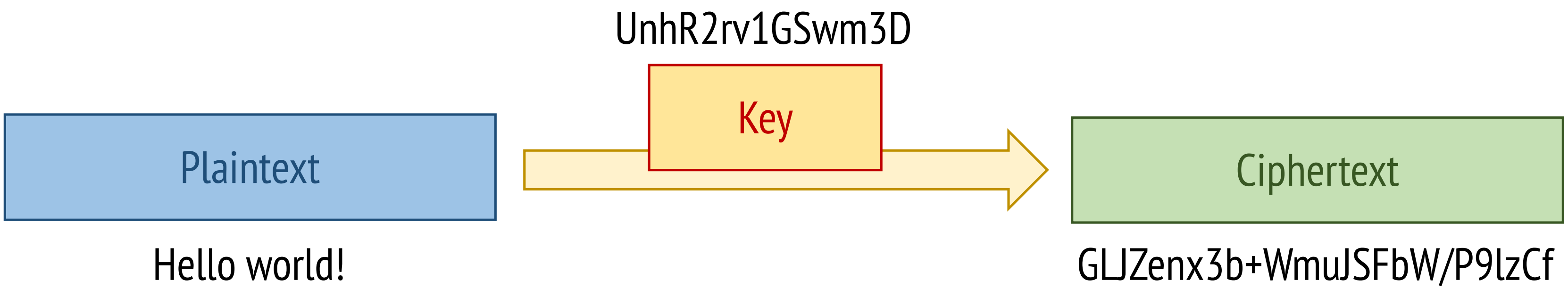
Overview

- Data security
  - Protecting the operating system partition
  - Protecting user data
  - Protecting user data in the cloud

Introduction

Symmetric Encryption

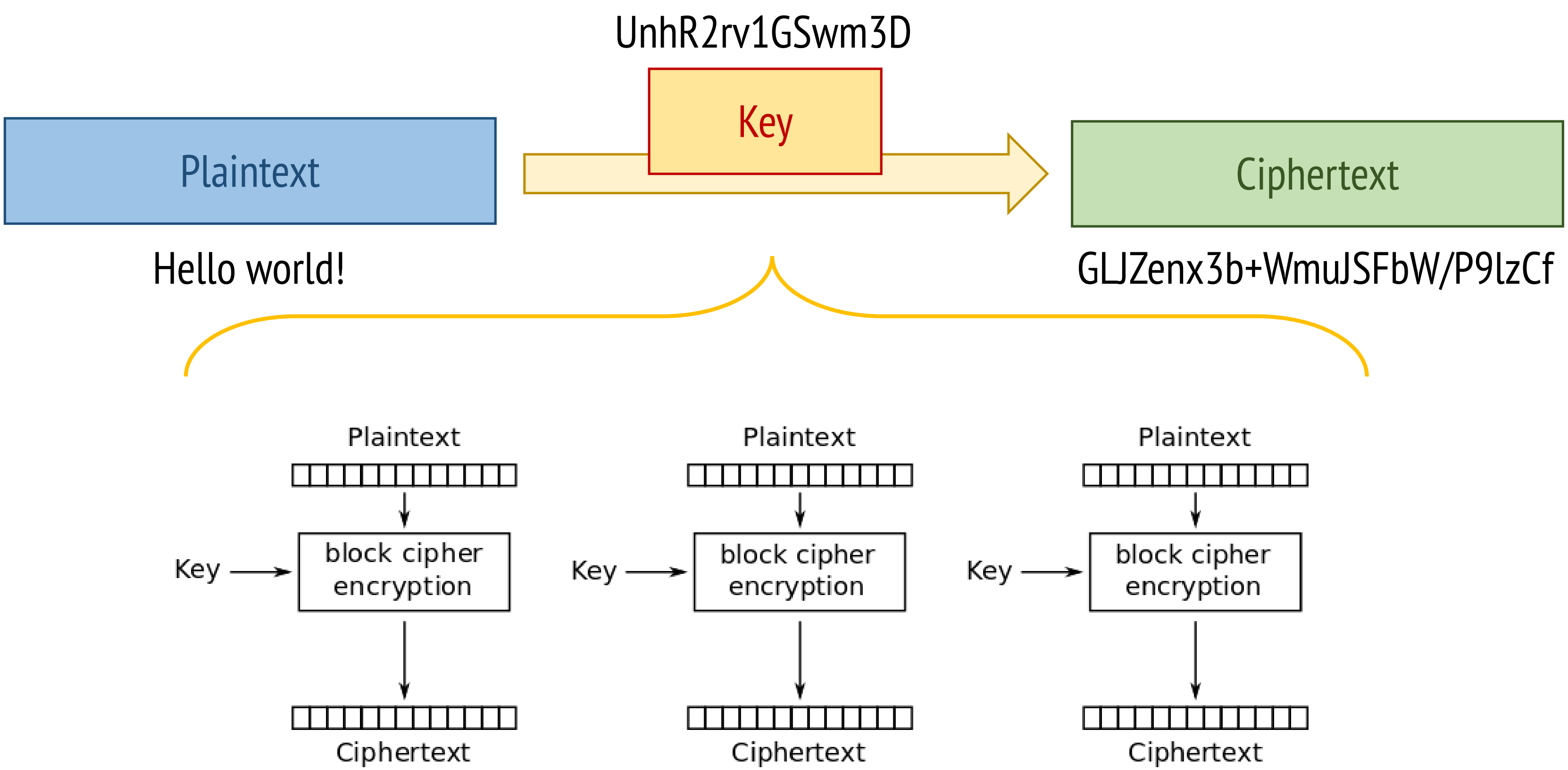
Encryption:



Symmetric Encryption

Key

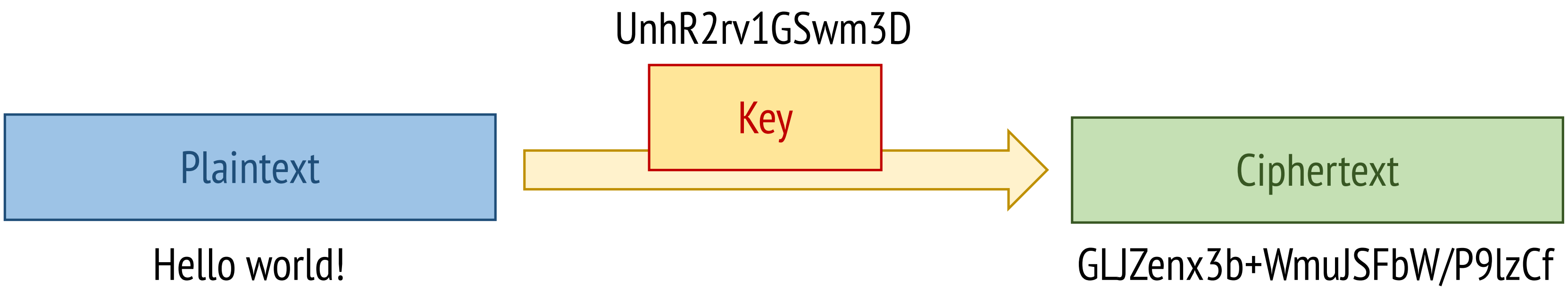
Encryption:



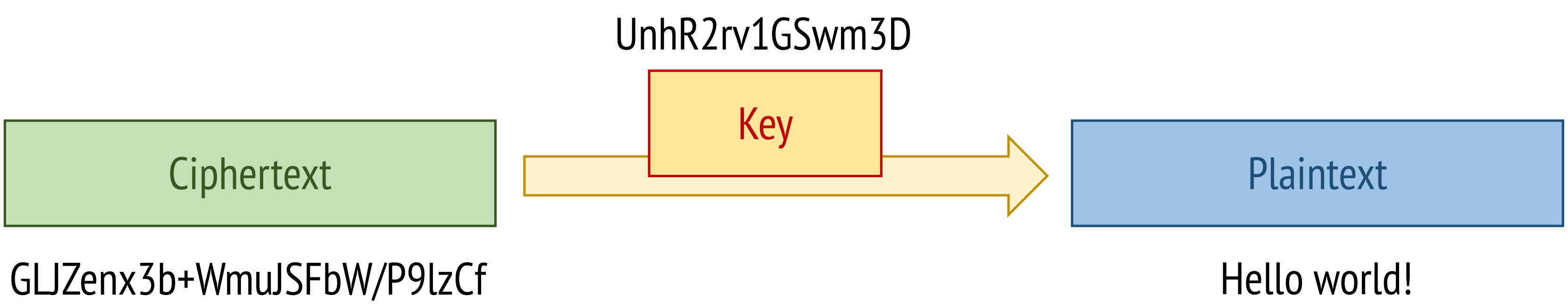
Electronic Codebook (ECB) mode encryption

Symmetric Encryption

Encryption:



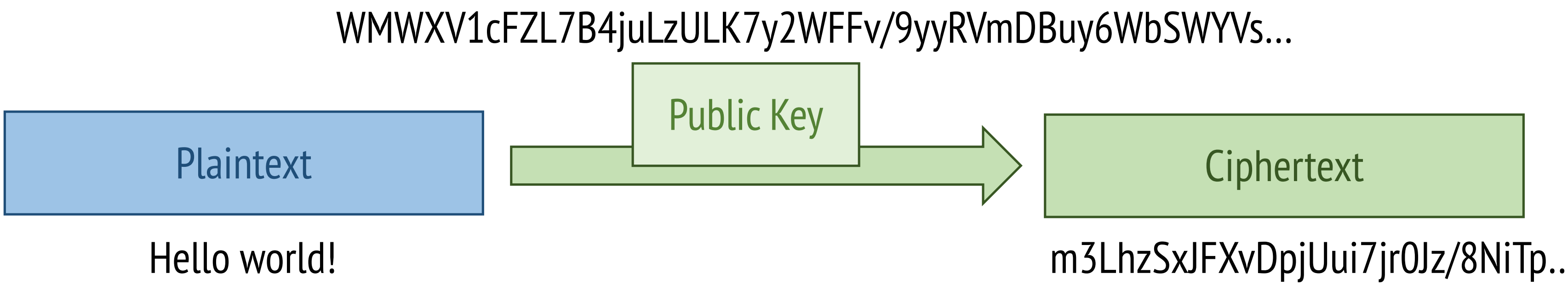
Decryption:



Public key encryption

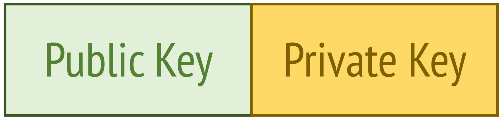
Public Key | Private Key

Encryption:

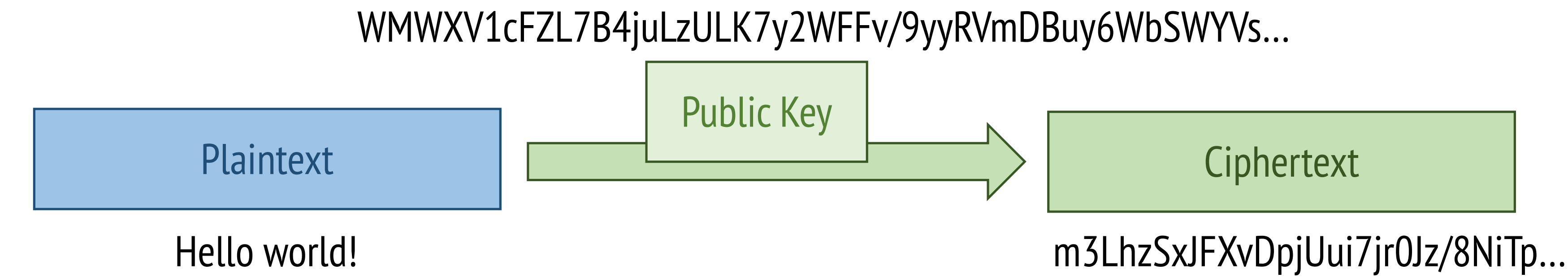


$$ciphertext \equiv plaintext^{public\_key} \pmod{n}$$

Public key encryption

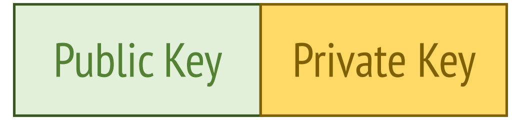


Encryption:

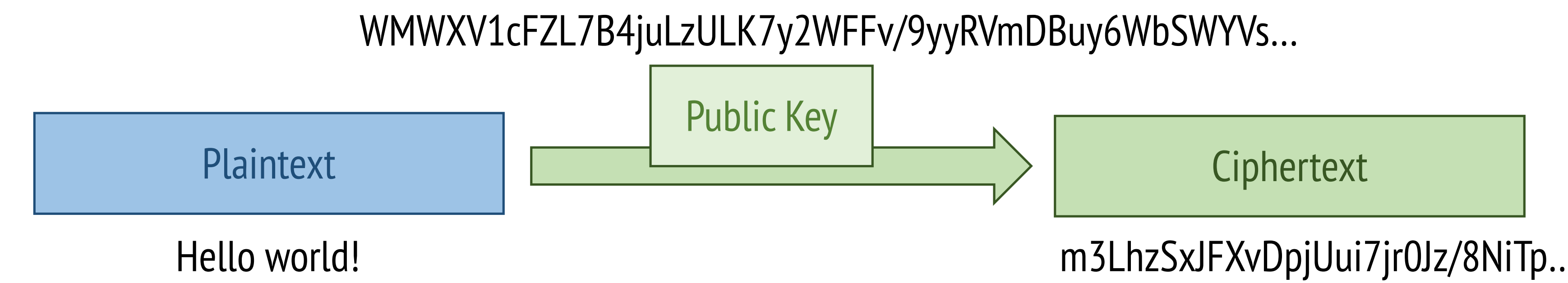


Decryption:

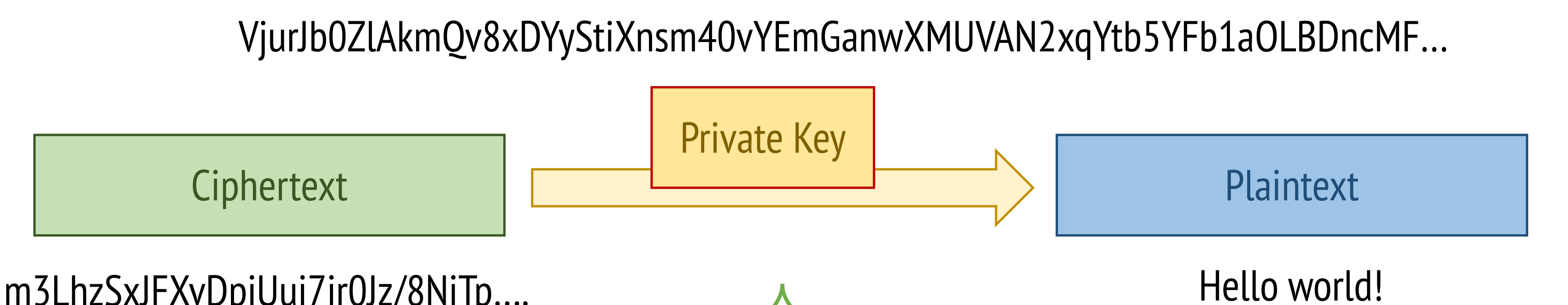
Public key encryption



Encryption:



Decryption:



$$ciphertext^{private\_key} \equiv plaintext \pmod{n}$$

Public key vs. symmetric key

Public key cryptography

- Anyone can encrypt messages and only the key owner can decrypt the ciphertext
- Public key requires longer keys
- The resulting longer ciphertexts are larger than the plaintext
- …

Symmetric key cryptography

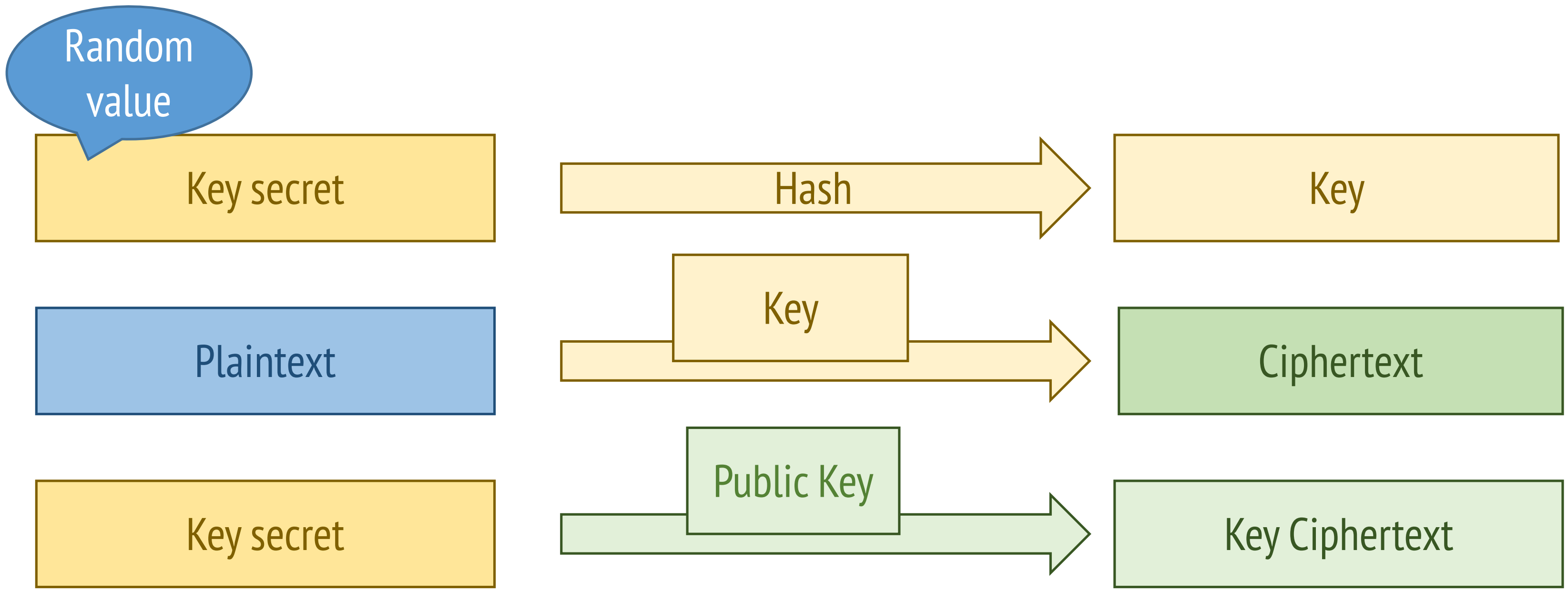
- The encryption/decryption key needs to be shared between parties
- Keys are relatively small
- Resulting ciphertext is about the same size as the plaintext
- …

Key encapsulation mechanisms (KEM)

KEMs are an efficient method to securely share symmetric keys with the help of public keys.

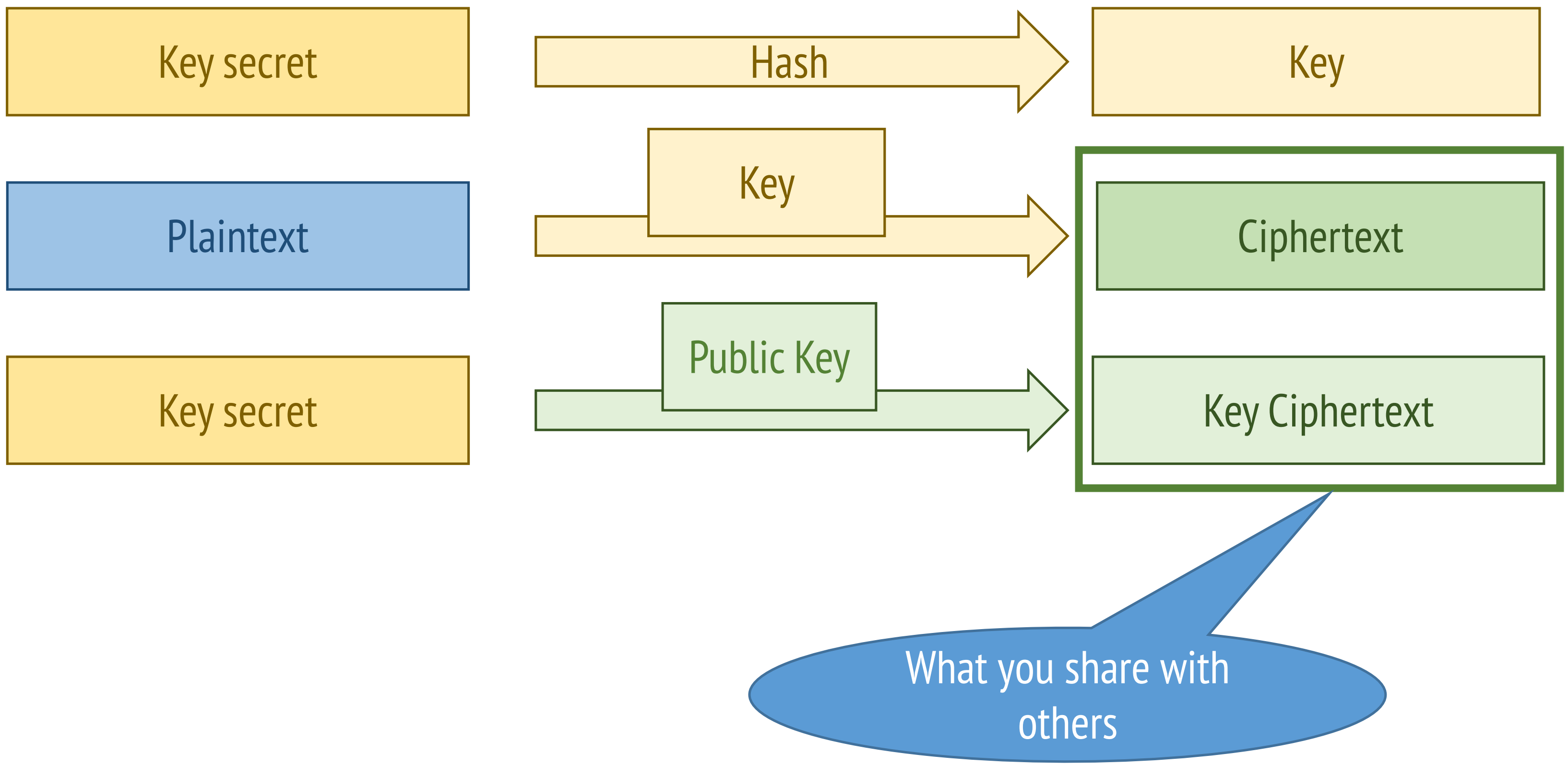
Key encapsulation mechanisms (KEM)

Encryption:



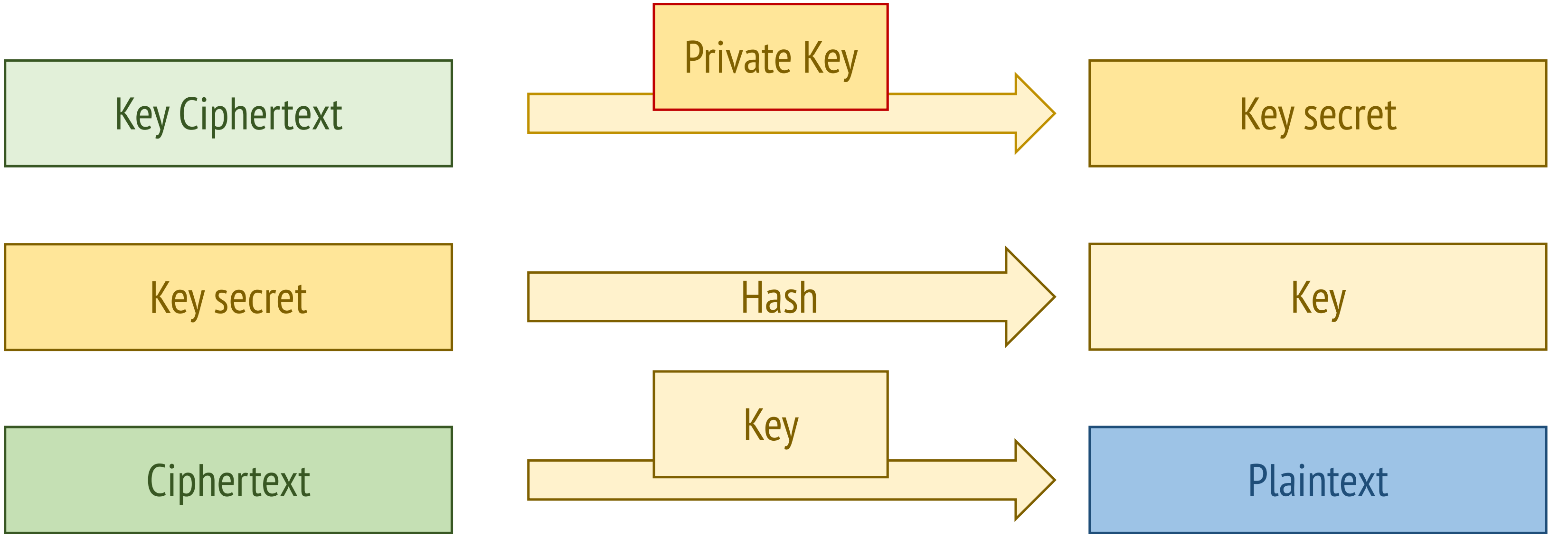
Key encapsulation mechanisms (KEM)

Encryption:



Key encapsulation mechanisms (KEM)

Decryption:



Key encapsulation mechanisms (KEM)

Advantages:

- Symmetric key has good entropy (output of hash function)
- Anybody can encrypt <plaintexts> for the private-key holder
- **Small overhead**

What can we protect?

- **Data at rest** is inactive data that is stored physically in any digital form e.g. files, databases, backups, but also swap
- **Data in use** is data being processed by a CPU or RAM.

What you get/don’t get from encryption?

Encryption does:

- Protect data while resting (i.e. your device is off)
- Protect data from apps who don’t have access to the keys (assuming sandboxing is used)
- Protect data from if un-authorised repairs are done (or device is stolen)

Encryption does not:

- Prevent data loss (it could actually make it easier).
- Make the system more resilient (quite the opposite: you will be more susceptible to DoS attacks).
- Data that has been decrypted in the volatile memory (RAM).

Challenges

Goal:

Complement the “trusted boot” with data confidentiality.

Challenges:

1a. How much information about data should be revealed?

2a. Who should be able access this information data?

…

1b. How to provide confidentiality to the **system-data** required to boot the system?

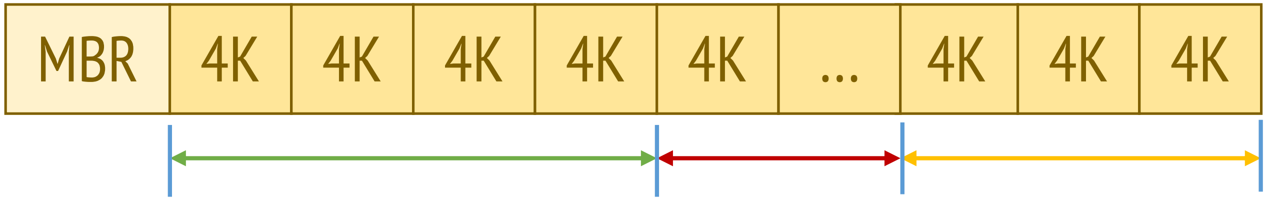
…

Types of data encryption

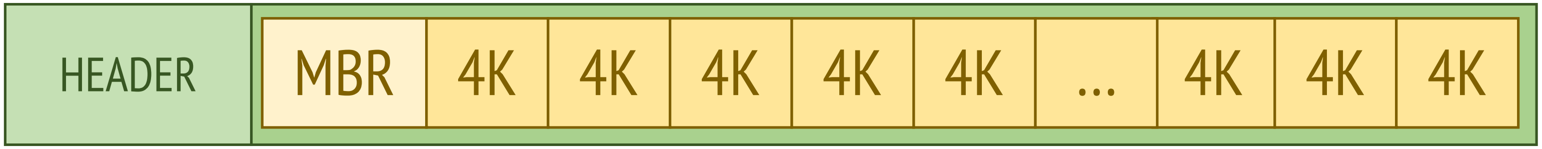
- Disk based
- File based

Disk based encryption

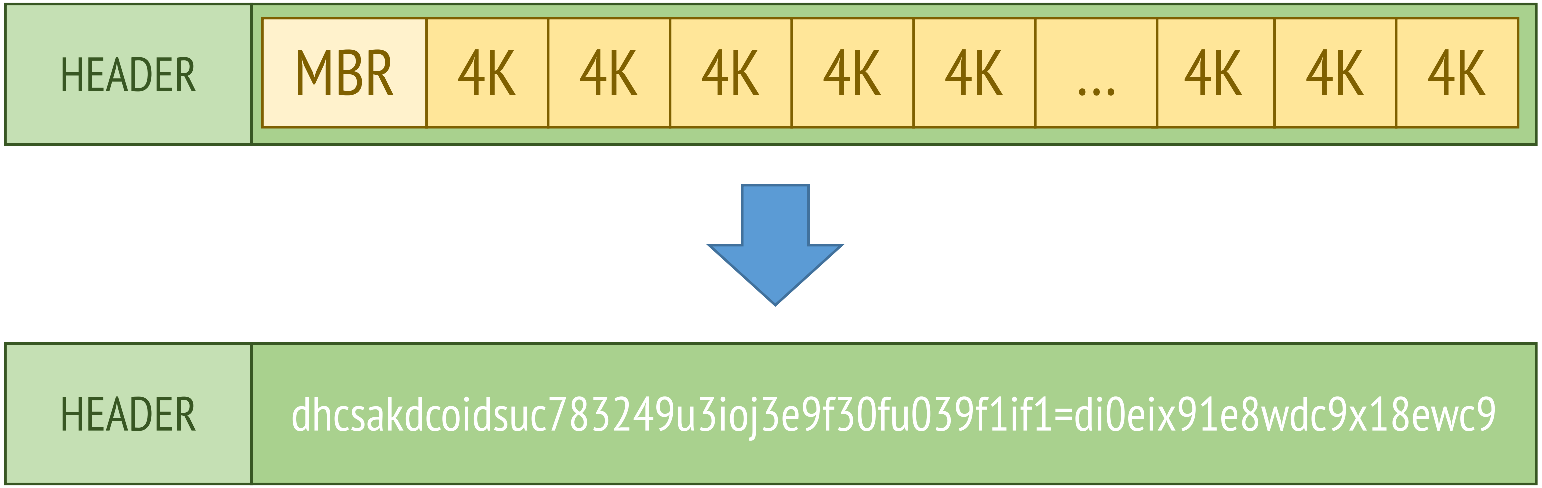
Partition:



Disk based encryption



Disk based encryption

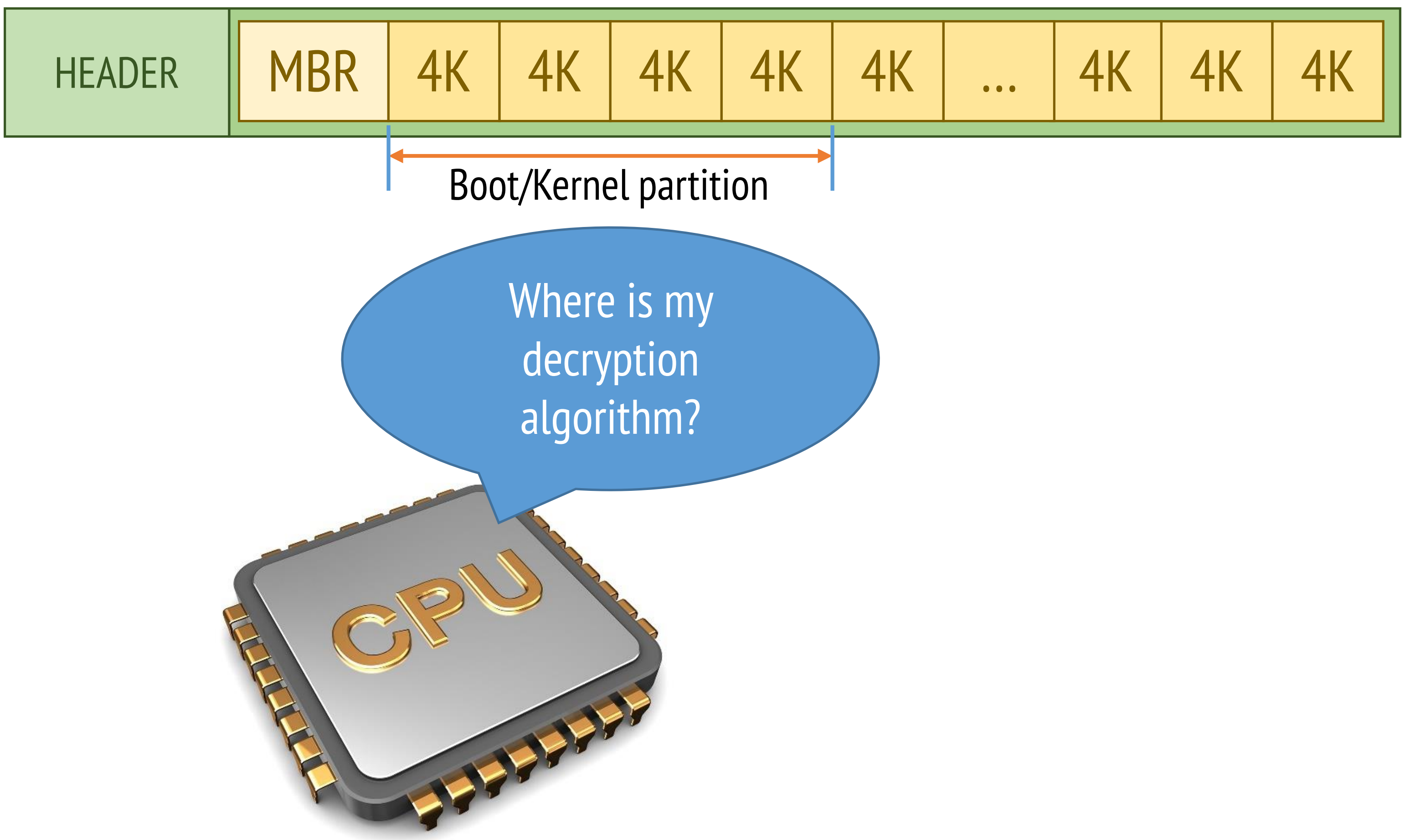


Disk based encryption

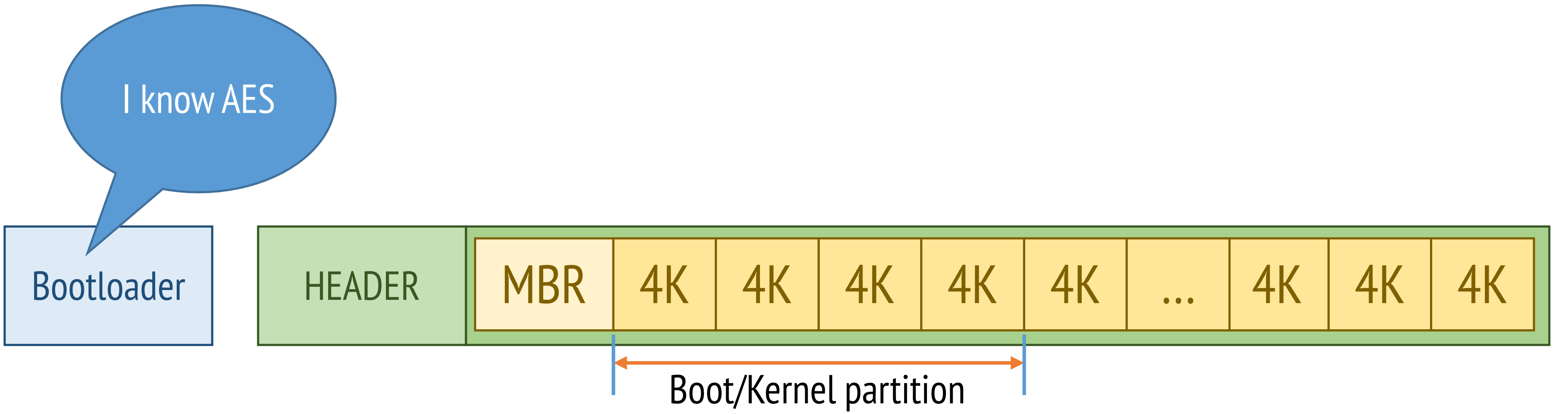
HEADER

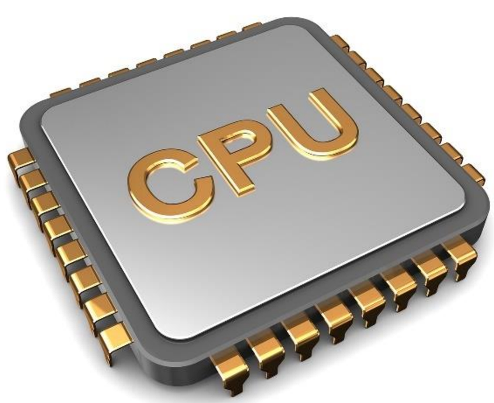
Type: LUKS/BitLocker/etc.
Cipher: AES
IV: ndef731o4jf92
Requirements: TPM
Comment: “main partition”
…

In practice challenges

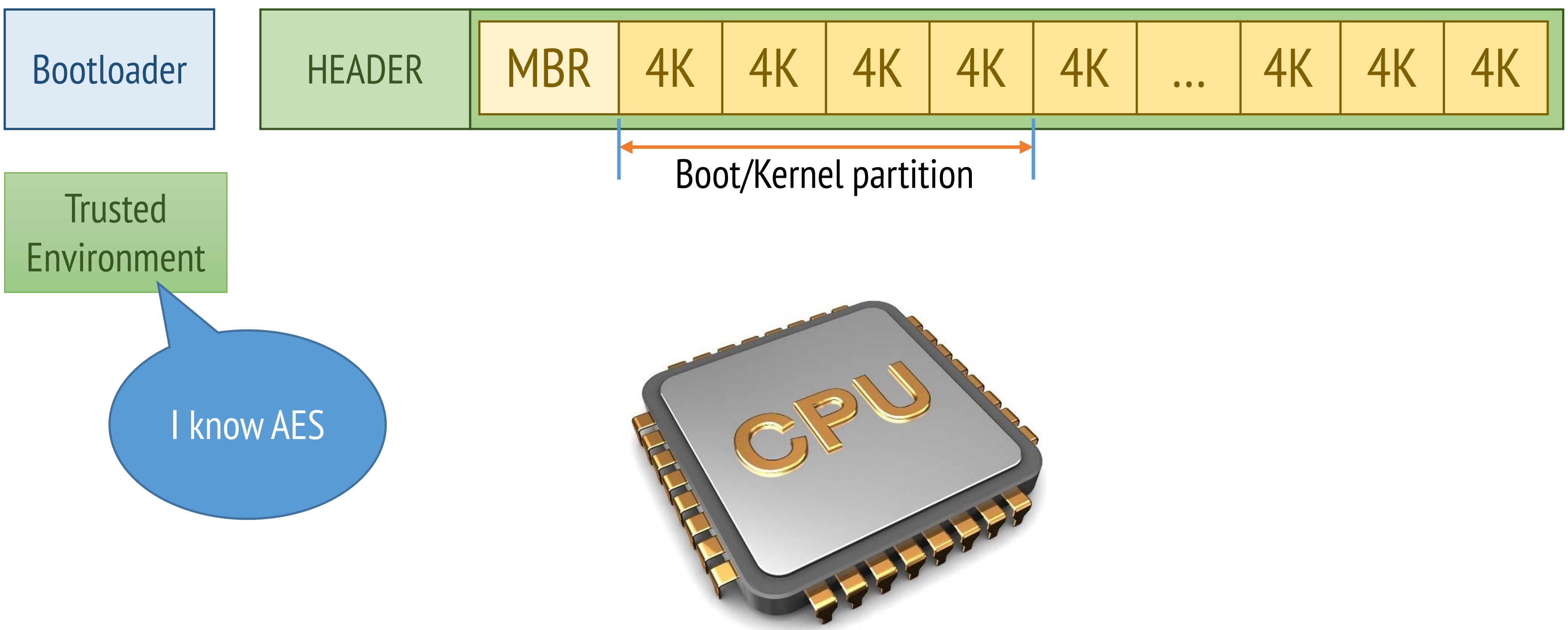


In practice challenges

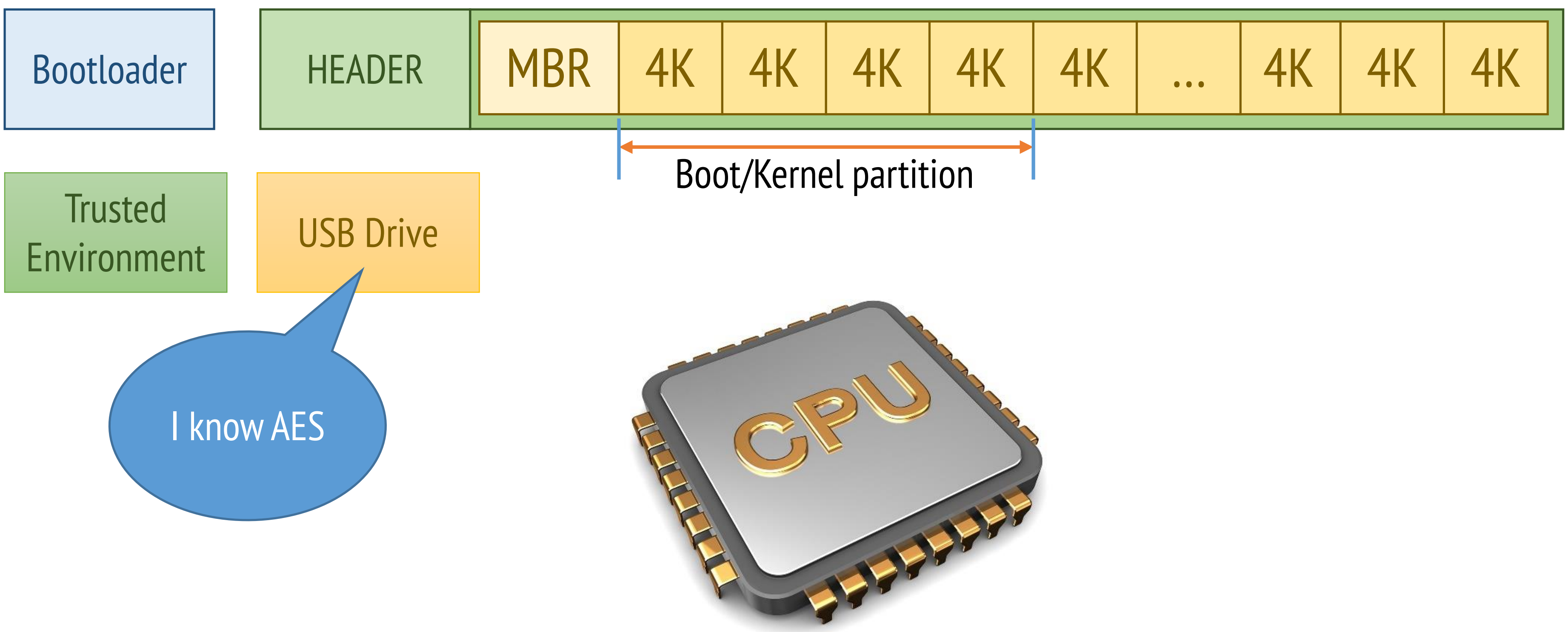




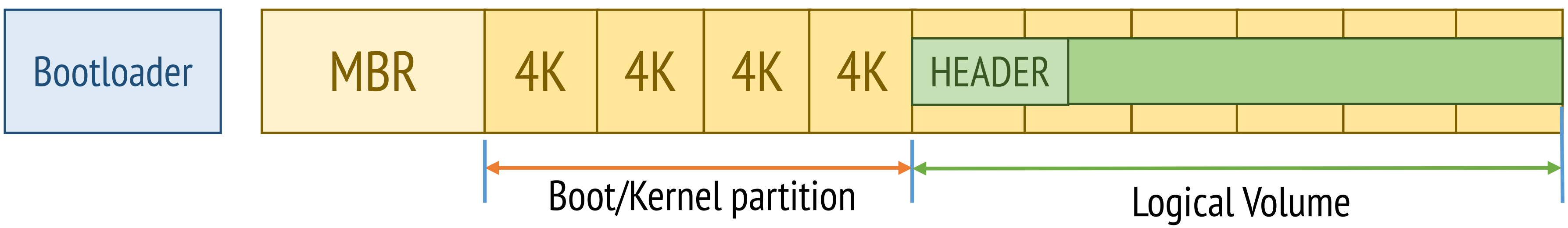
In practice challenges



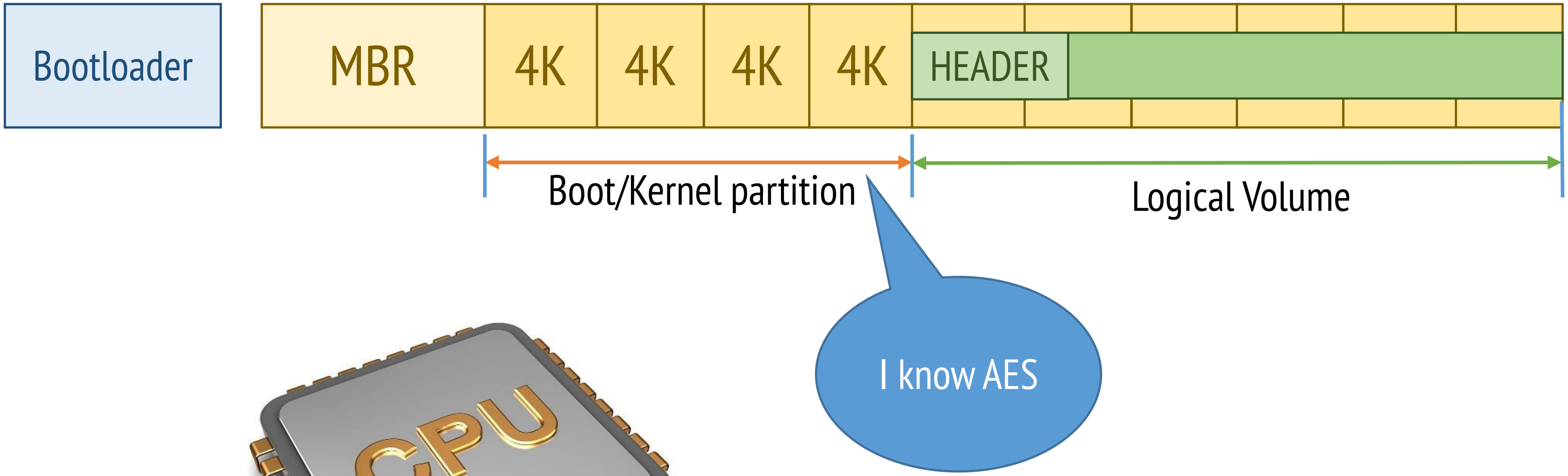
In practice challenges

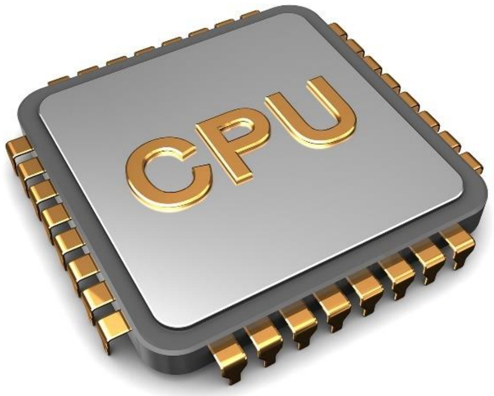


In practice challenges



In practice challenges





Hold on, are we not missing something important?

Hold on, are we not missing something important?

Right… the **encryption key.**

Storing the key

On a USB stick:

- Easy
- Requires USB to be accessible to the system
- Vulnerable to stealing

In the TPM (or TEE)

- More difficult to set up
- Transparent
- Protected from stealing

Storing the key

On a SmartCard:

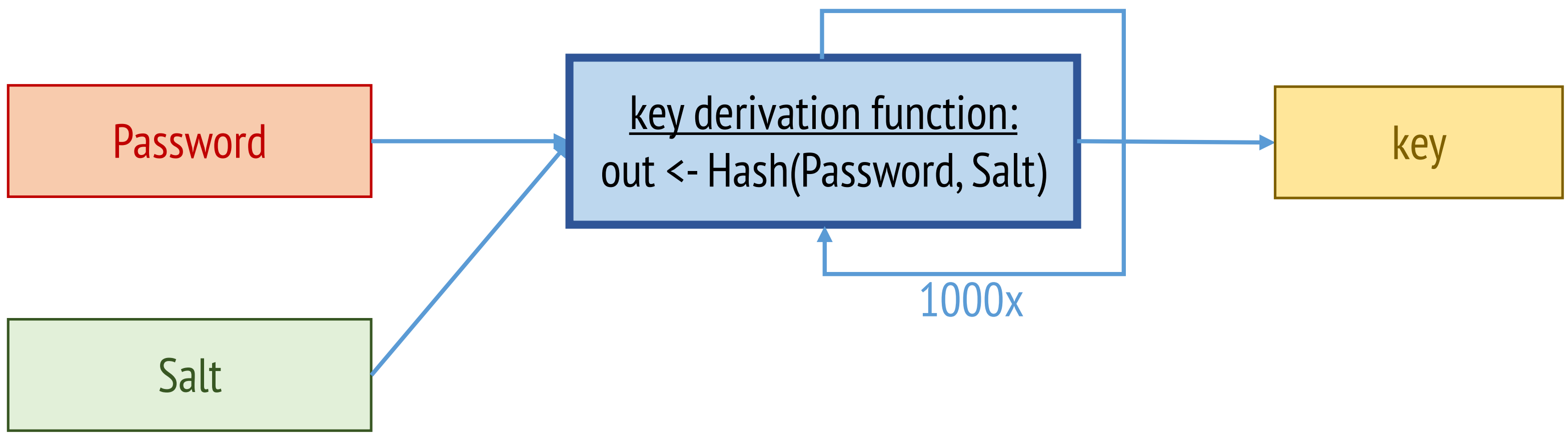
- Difficult to set up (e.g. requires special hardware)
- Requires presence of the card
- Protected from stealing

Deriving the key from a password

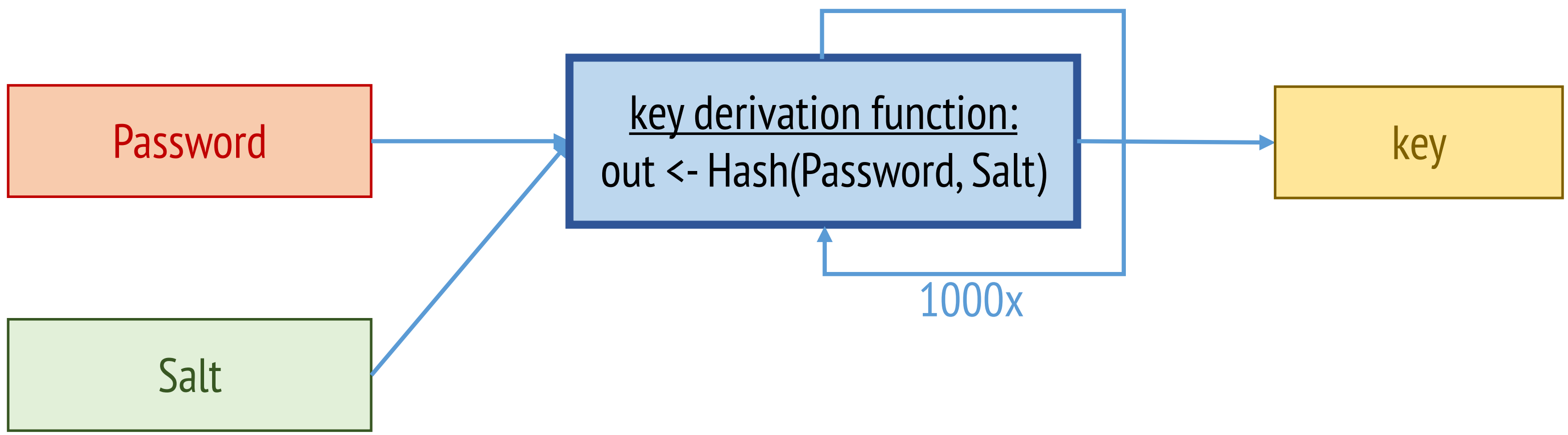
Challenge:

- Human generated passwords have low entropy!

Key derivation functions



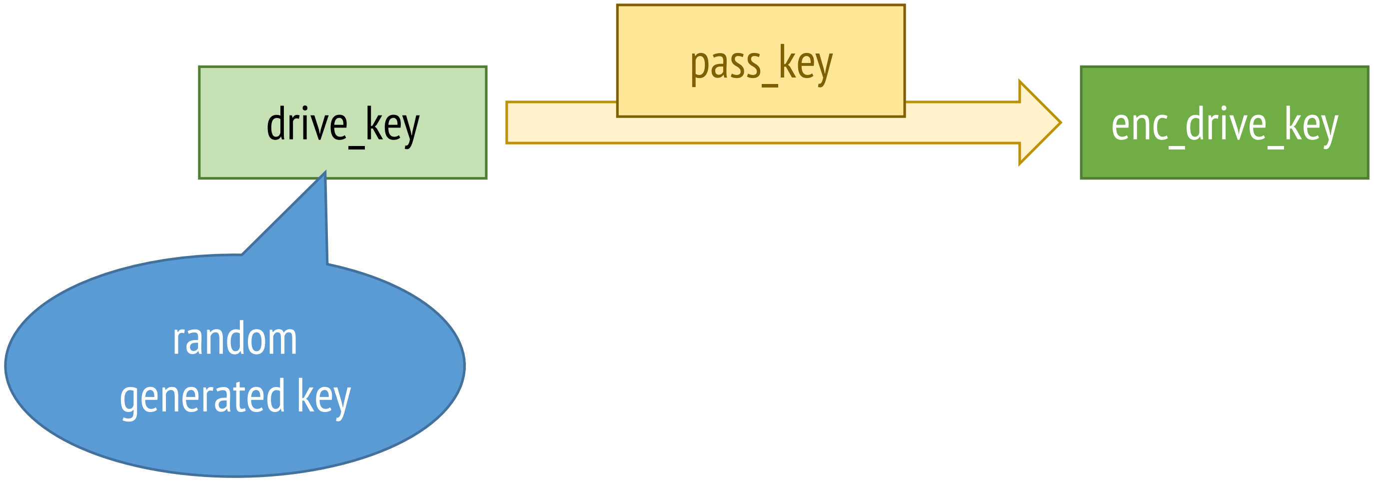
Key derivation functions



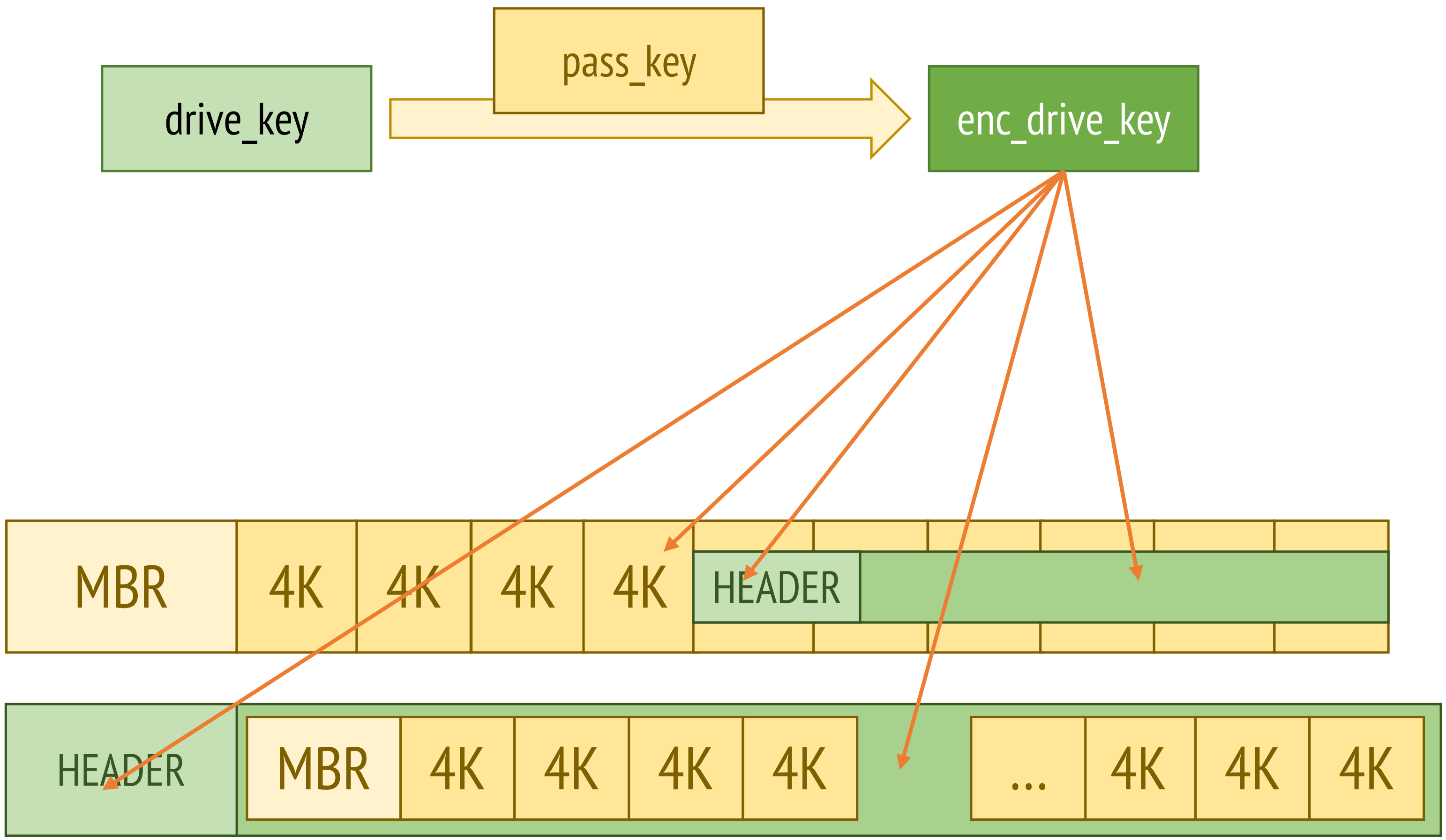
The key has sufficient entropy!

What if I want to change my password?
Do I have to re-encrypt the whole drive?

Key derivation functions



Key derivation functions



Decryption

Derived key decryption process:

- Load enc_drive_key from the drive
- Derive key from password
- Decrypt key
- Decrypt drive

Decryption

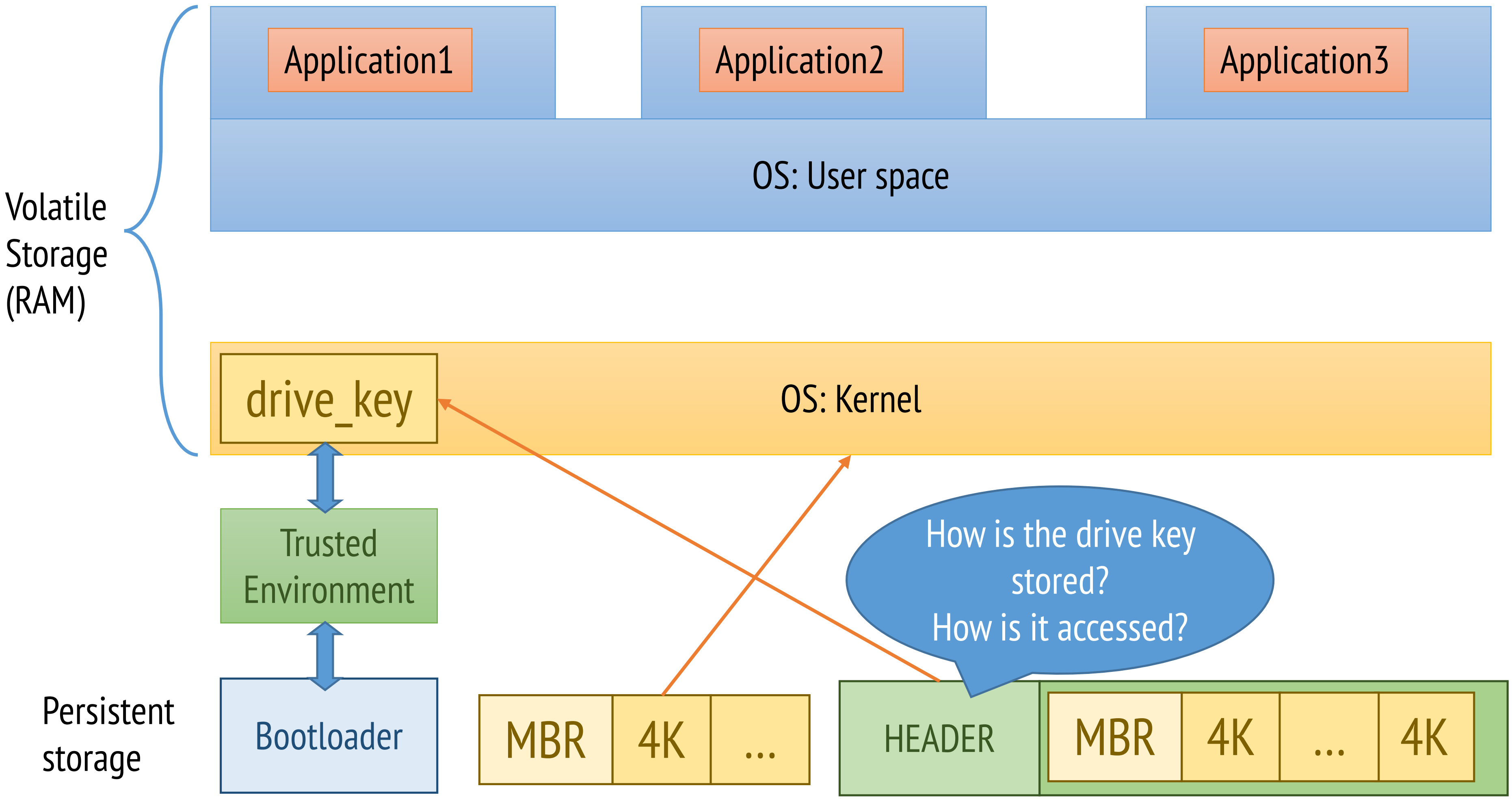
Stored key decryption:

1. Load key from USB
2. Decrypt drive

Or:

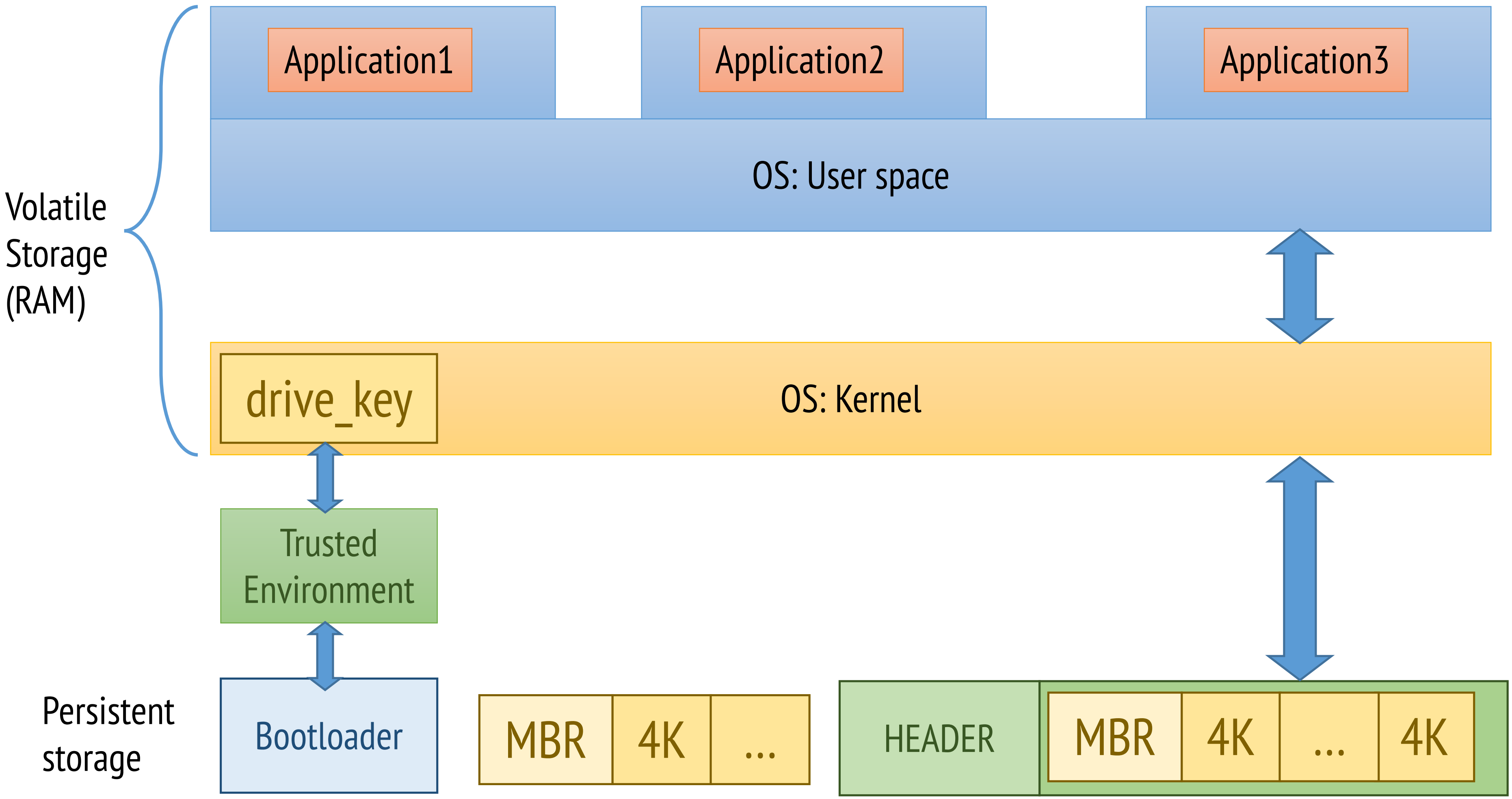
1. Load key from enc_drive_key
2. Load key from USB/SmartCard/TEE/TPM
3. Decrypt enc_drive_key
4. Decrypt drive

Usage

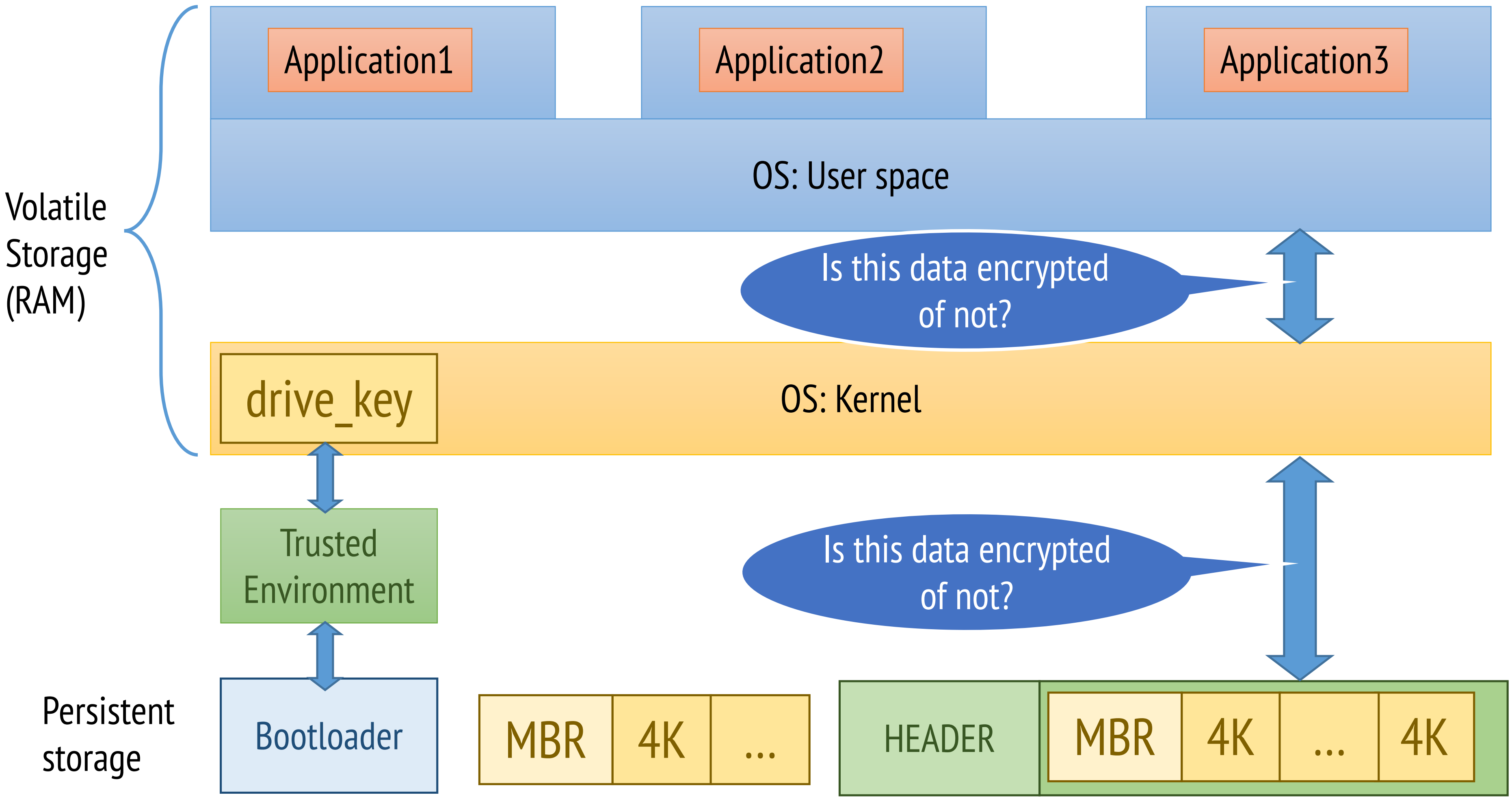


Volatile Storage (RAM)
Application1
Application2
Application3
OS: User space
drive_key
OS: Kernel
Trusted Environment
How is the drive key stored?
How is it accessed?
Persistent storage
Bootloader
MBR
4K
...
HEADER
MBR
4K
...
4K

Usage

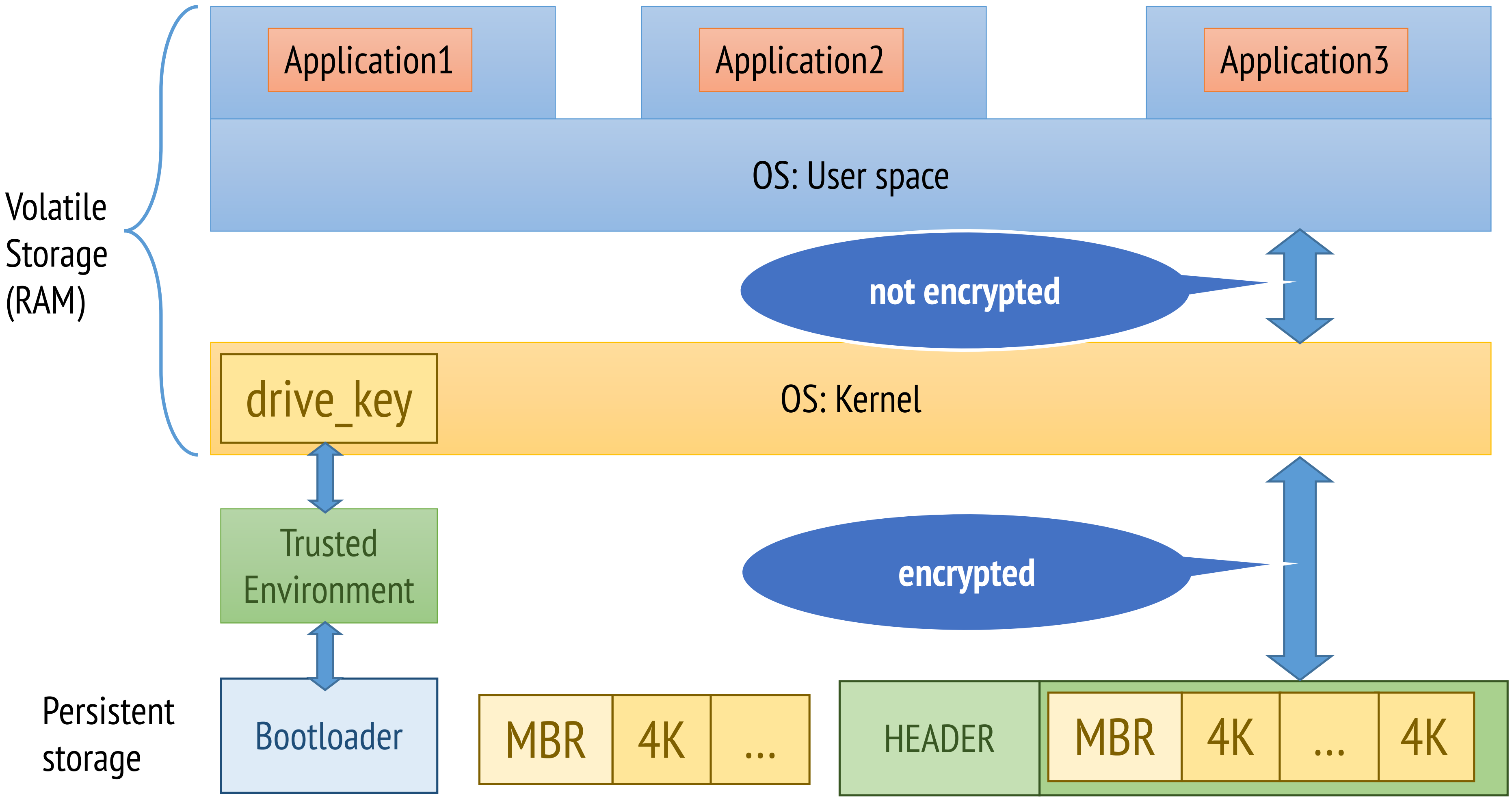

Volatile
Storage
(RAM)
Application1
Application2
Application3
OS: User space
drive_key
OS: Kernel
Trusted
Environment
Persistent
storage
Bootloader
MBR
4K
...
HEADER
MBR
4K
...
4K

Usage



Application1
Application2
Application3
OS: User space
Volatile
Storage
(RAM)
Is this data encrypted
of not?
drive_key
OS: Kernel
Trusted
Environment
Is this data encrypted
of not?
Persistent
storage
Bootloader
MBR
4K
...
HEADER
MBR
4K
...
4K

Usage



Application1
Application2
Application3
OS: User space
Volatile
Storage
(RAM)
not encrypted
drive_key
OS: Kernel
Trusted
Environment
encrypted
Persistent
storage
Bootloader
MBR
4K
...
HEADER
MBR
4K
...
4K

Real implementations

BitLocker

- **Transparent operation mode**: uses the capabilities of TPM hardware to provide for a transparent user experience by sealing it on the TPM chip.

- **User authentication mode**: the user has to authenticate before decryption starts.

- **USB/ smartcard key mode**: the user must insert a USB device that contains a the key into the computer.

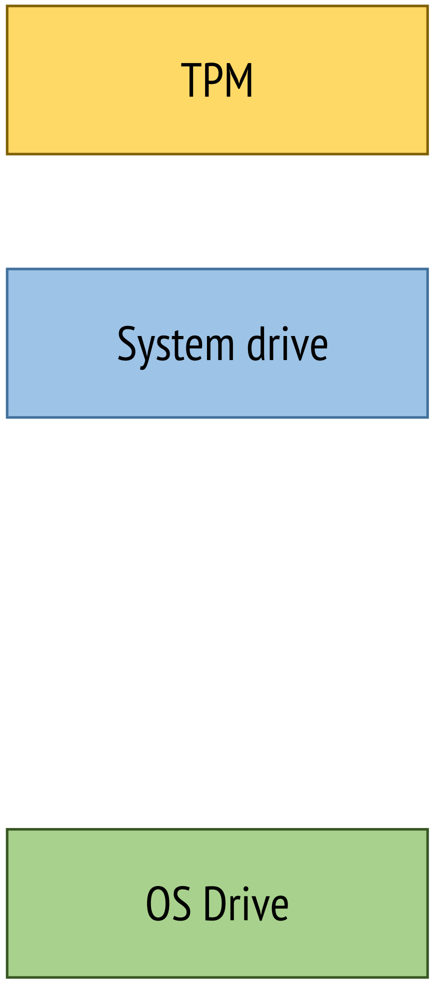
BitLocker keys

Keys:

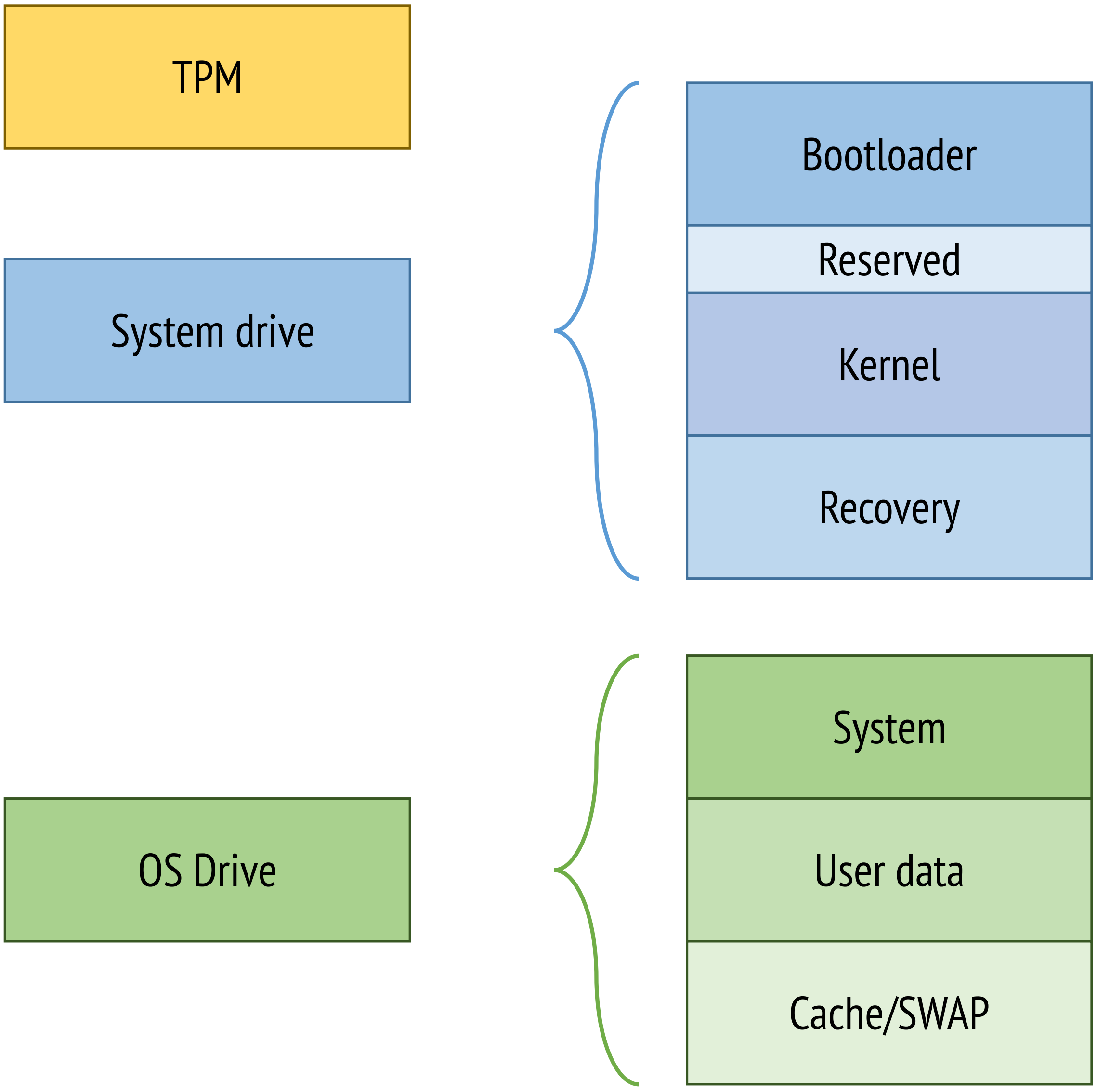
- Data Encryption Key (DEK): the drive generates the DEK and it never leaves the device. It is stored in an encrypted format at a random location on the drive. If the DEK is changed or erased, data encrypted using the DEK is irrecoverable.

- Authentication Key (AK): the key used to unlock data on the drive. A hash of the key is stored on drive and requires confirmation to decrypt the DEK.

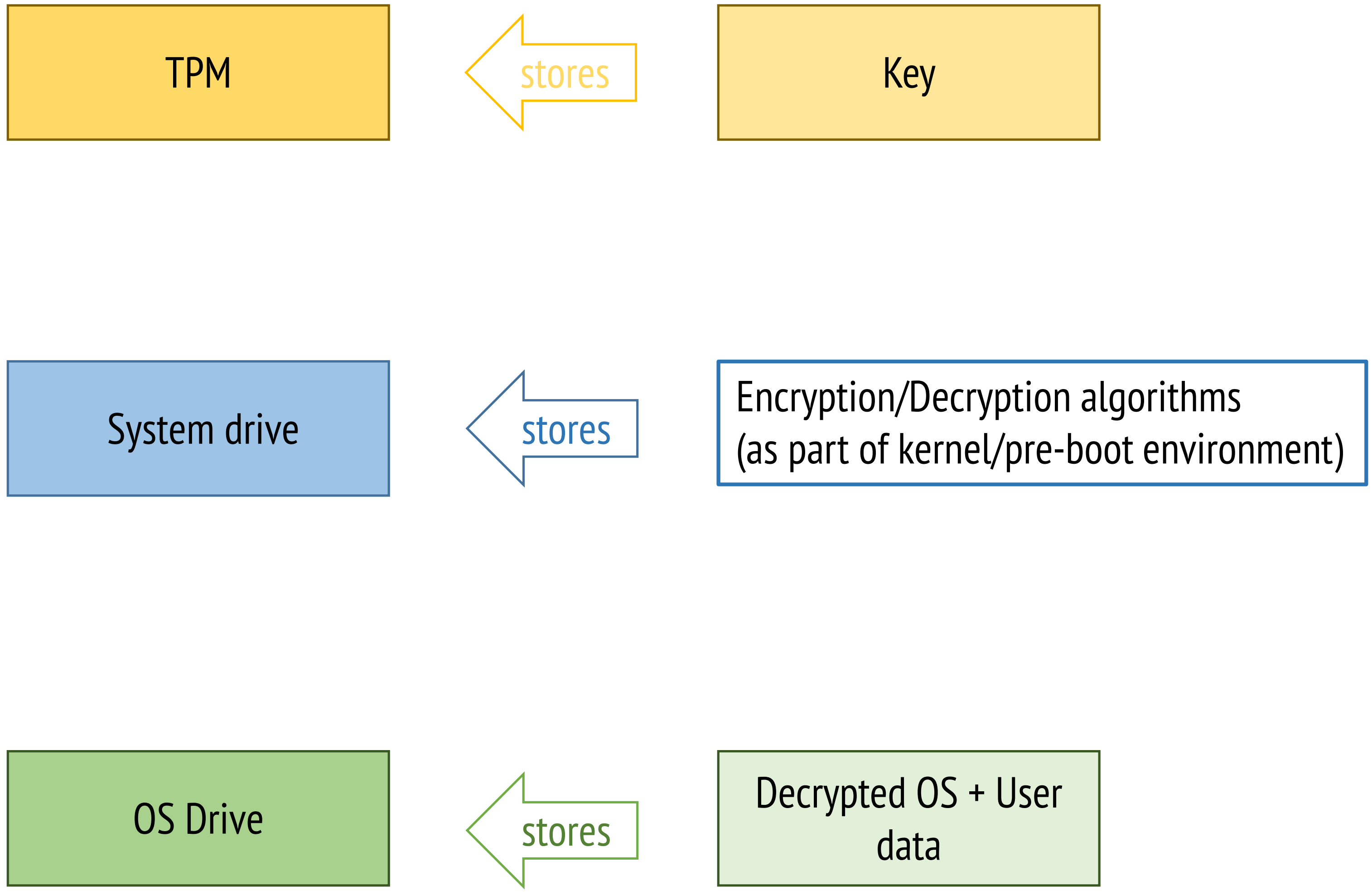
BitLocker



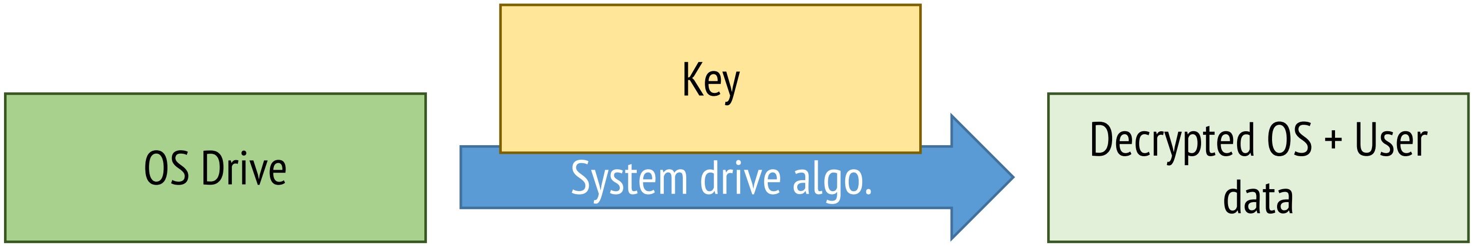
BitLocker


TPM
System drive
OS Drive
Bootloader
Reserved
Kernel
Recovery
System
User data
Cache/SWAP

BitLocker


TPM
stores
Key
System drive
stores
Encryption/Decryption algorithms
(as part of kernel/pre-boot environment)
OS Drive
stores
Decrypted OS + User data

BitLocker



TPM extra protection

The **key** is sealed inside the TPM's memory
The **key** is only released if early boot files appear to be unmodified

Principles used

- Simple design
- Assume secrets not safe (e.g. the key is sealed inside the TPM)
- Make security usable (e.g. transparent BitLocker mode vs User authentication mode)
- ….

Android 5.0 (with Linux kernel & dm-crypt)

dm-crypt:

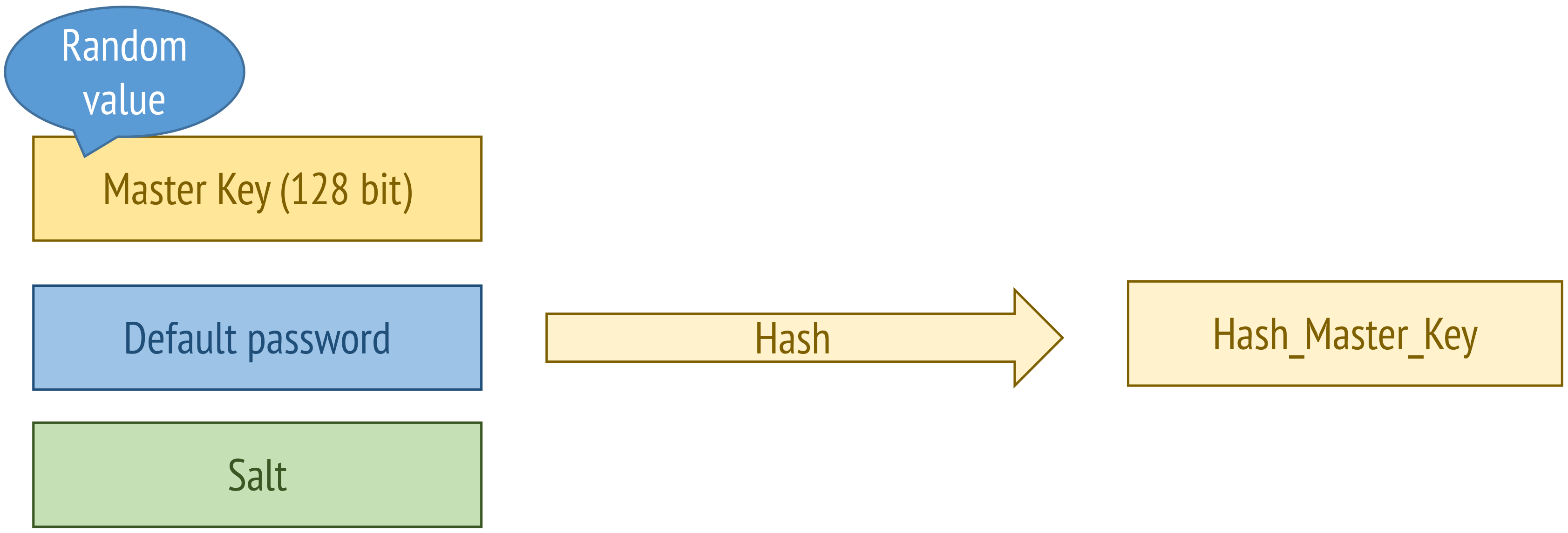
- Kernel module (runs in kernel space)
- Provides transparent disk encryption
- Supports the kernel only keys (i.e. logon keys)
- Uses cryptographic routines from the kernel’s Crypto API

Android 5.0 (with Linux kernel & dm-crypt)

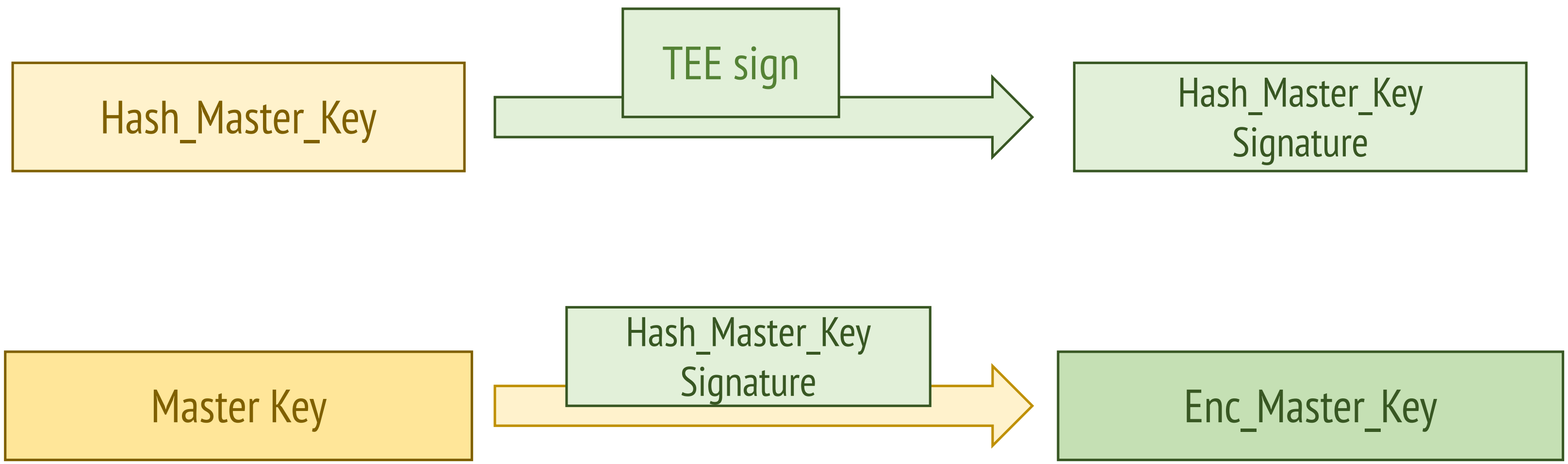
Android user authentication methods:

- default
- PIN
- password
- pattern

Android 5.0 (with Linux kernel & dm-crypt)



Android 5.0 (with Linux kernel & dm-crypt)

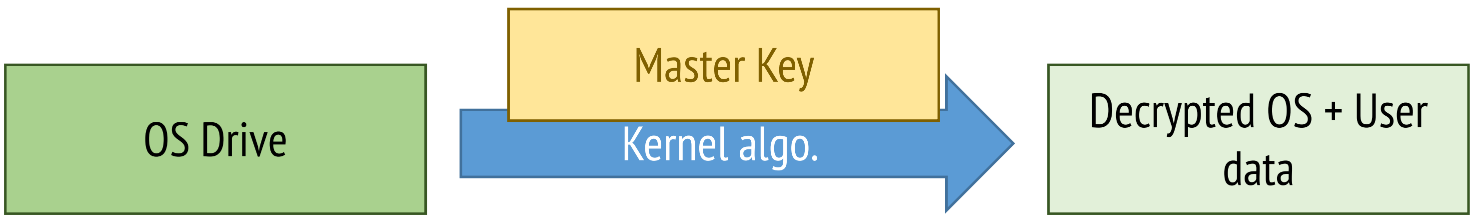


Android 5.0 (with Linux kernel & dm-crypt)

Which key am I using to decrypt my data?

Android 5.0 (with Linux kernel & dm-crypt)

Which key am I using to decrypt my data?



Android 5.0 (with Linux kernel & dm-crypt)

What happens to the Master Encryption key when I change my password, i.e. Default password?

Android 5.0 (with Linux kernel & dm-crypt)

What happens to the Master Encryption key when I change my password, i.e. Default Password?

Regenerate | Hash_Master_Key Signature | using | Master Key / New password / Salt?

Does the salt change?

Android 5.0 (with Linux kernel & dm-crypt)

What happens to the Master Encryption key when I change my password, i.e. Default Password?

Regenerate **Hash_Master_Key Signature** using

- Master Key
- New password
- New salt

Individual research

What are the security implications of using a public, hardcoded and known value for the “default password” in Android 5.0?

How does it compare to not using encryption at all?

Explain your answers by referring to security aspects (confidentiality, authentication…), and difficulty of use.

Advantages/disadvantages of full disk encryption

Full disk encryption

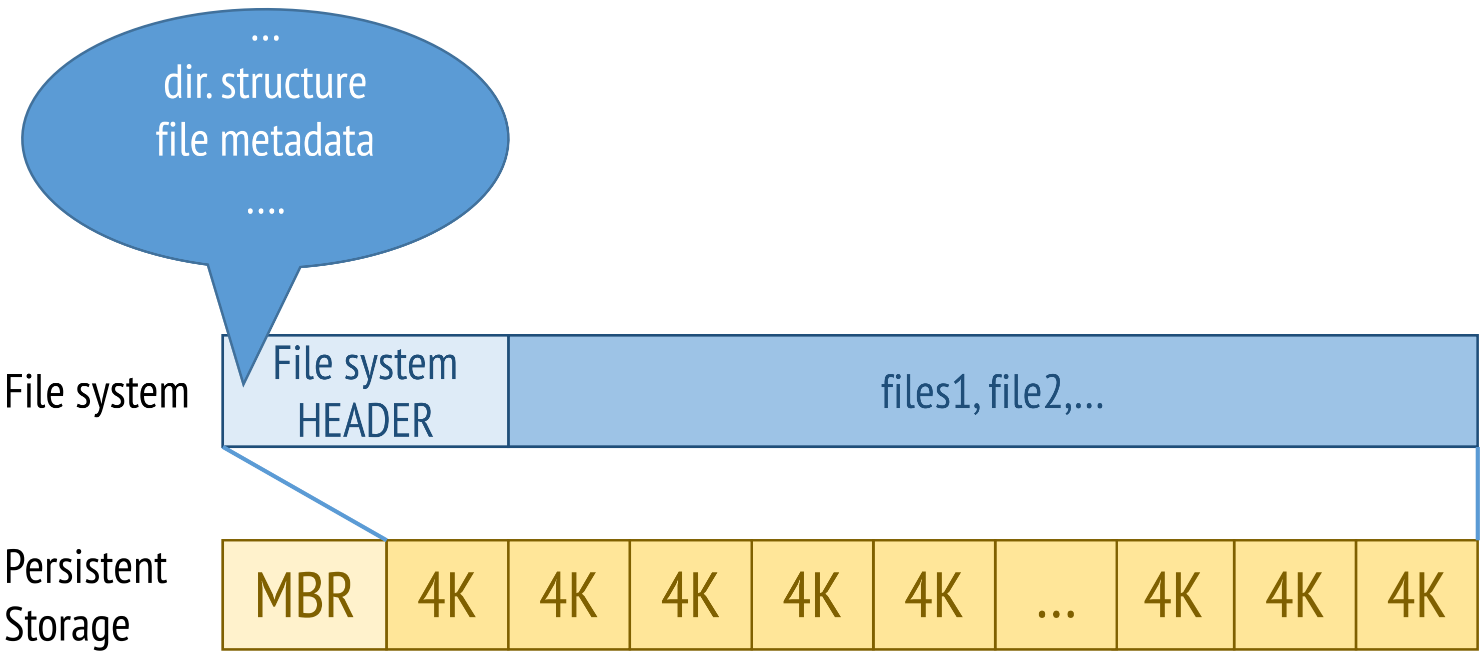
- Simple design: generally only one key is used.
- Protects filesystem meta data e.g. directory structure, file names, modification timestamps.
- If the key is compromised, the attacker has access to all files.

File based encryption

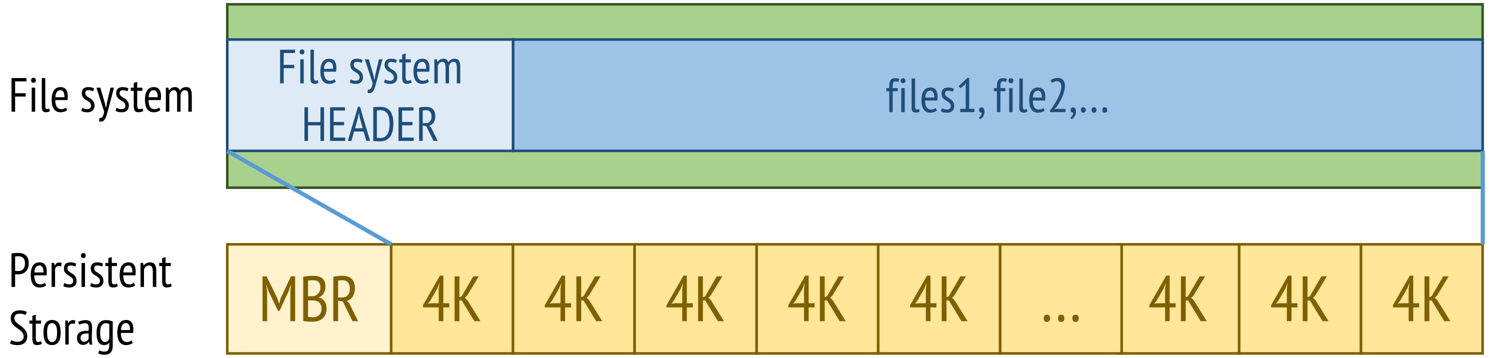
Components

- File contents
- File metadata
- Memory storage
- Disk storage
- Access control (type, user)

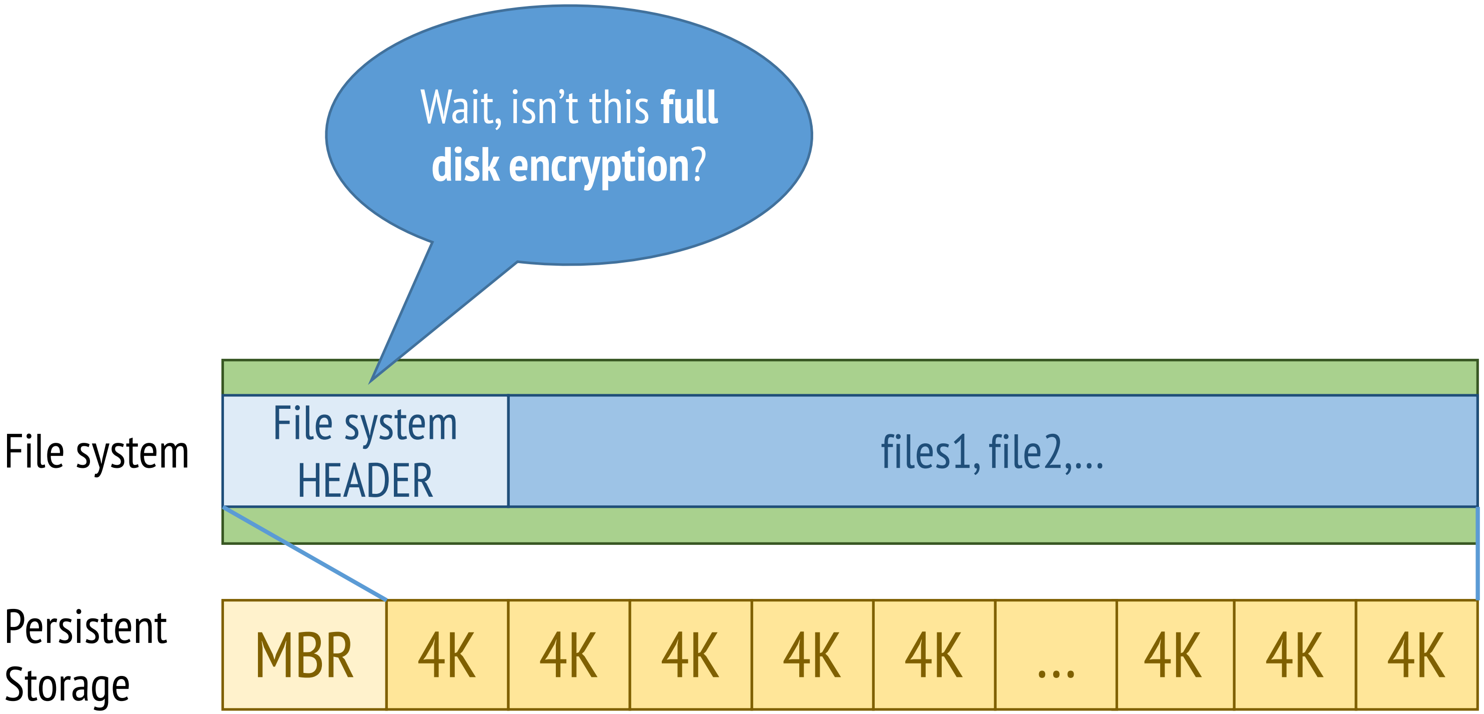
File based encryption



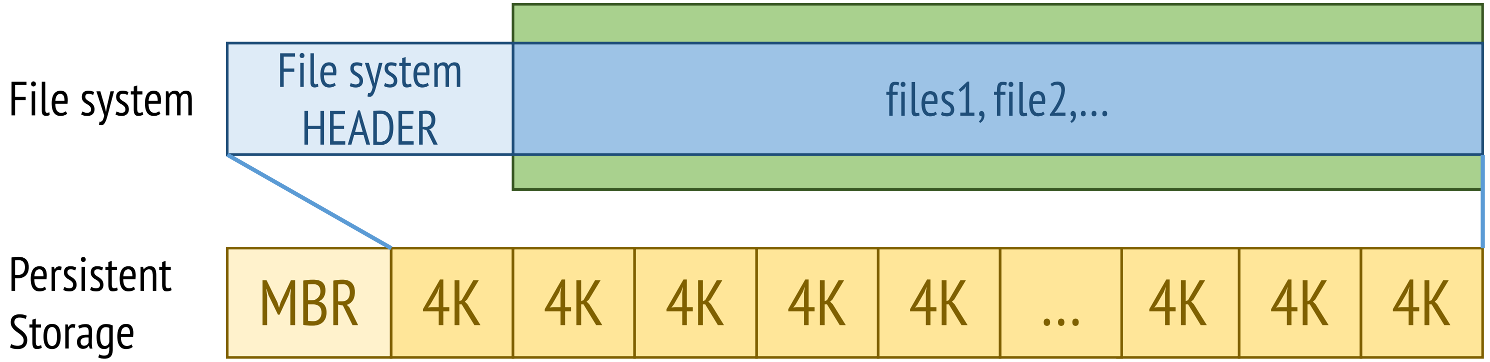
File based encryption



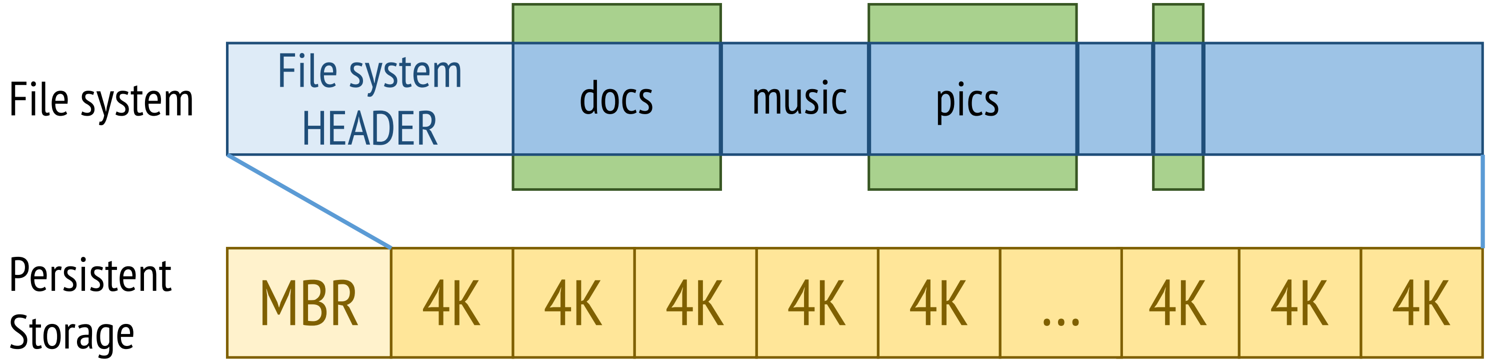
File based encryption



File based encryption



File based encryption

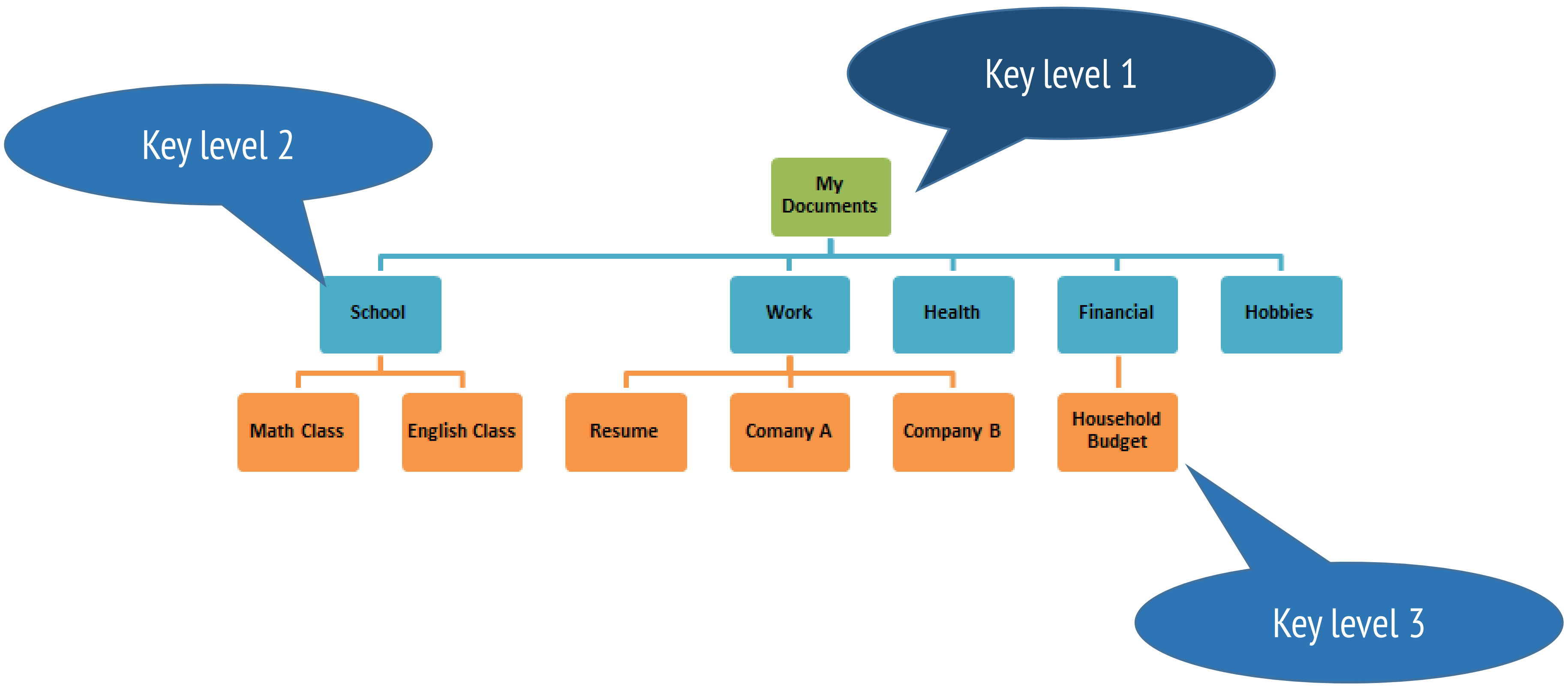


This is pretty cool. How do we do it?

Try to map a key structure to the file system structure.

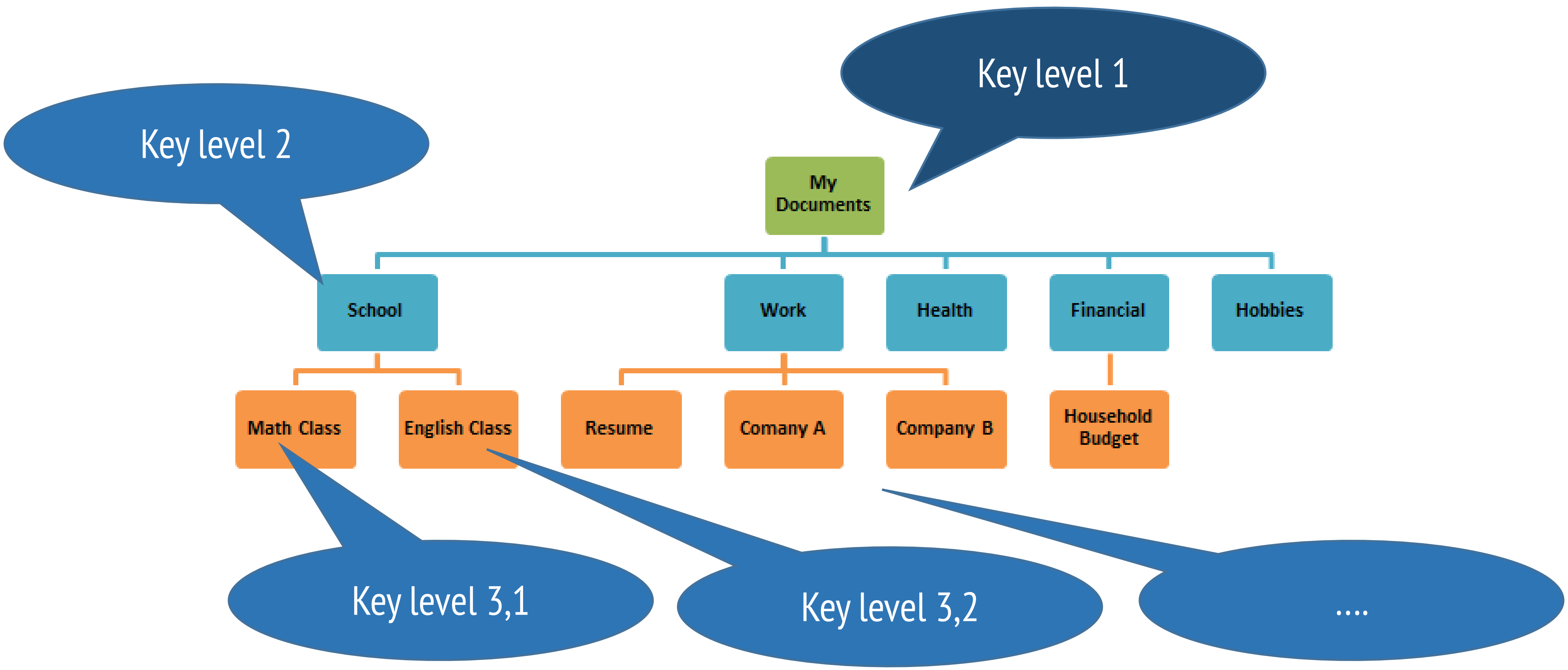
This is pretty cool. How do we do it?

Try to map a key structure to the file system structure.



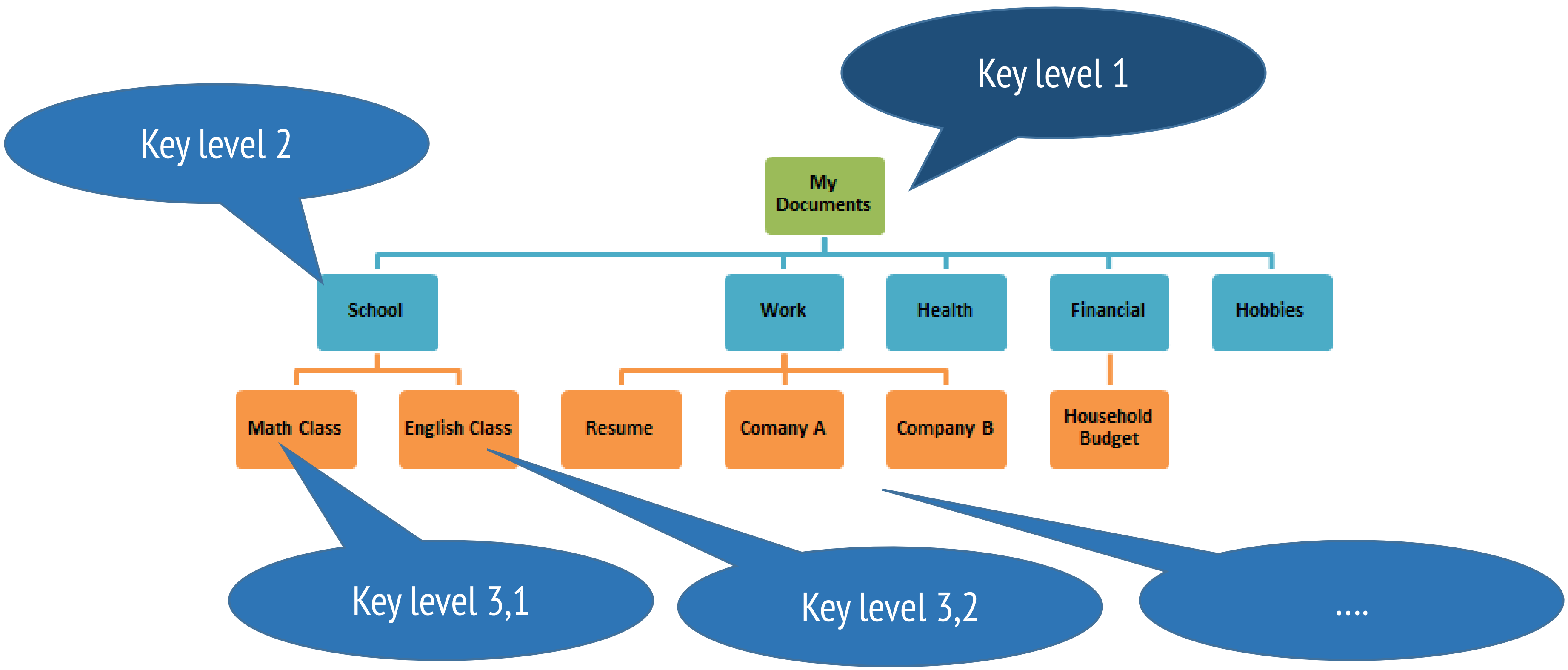
This is pretty cool. How do we do it?

Try to map a key structure to the file system structure.



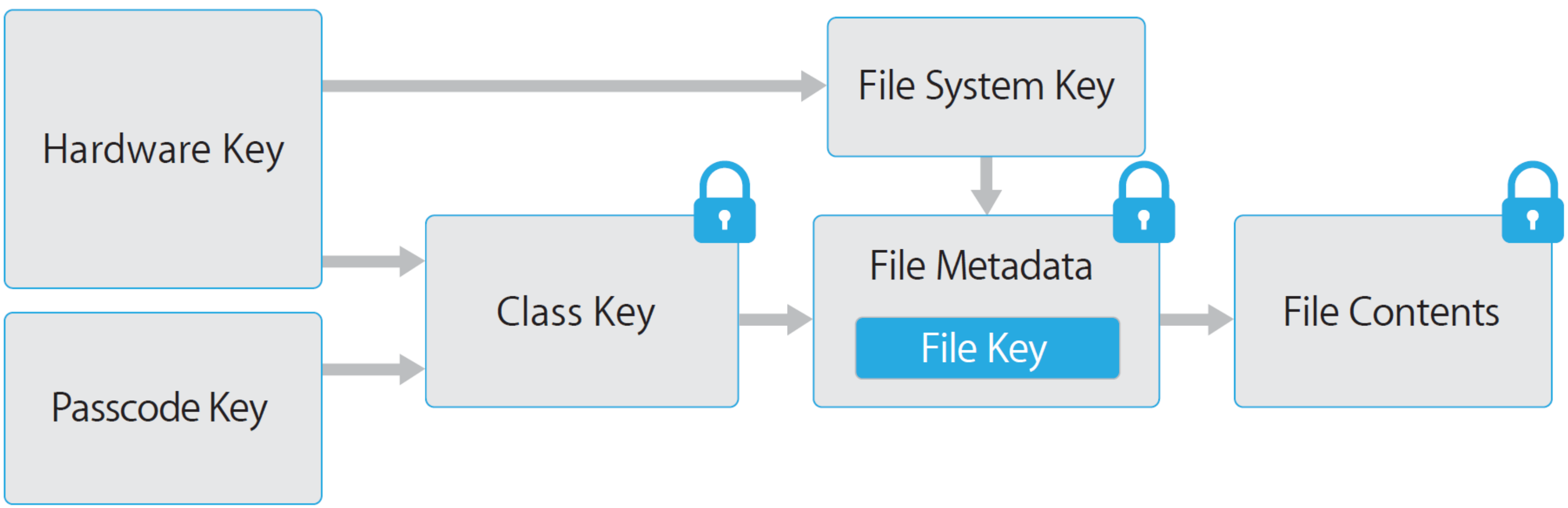
This is pretty cool. How do we do it?

Try to map a key structure to the file system structure.



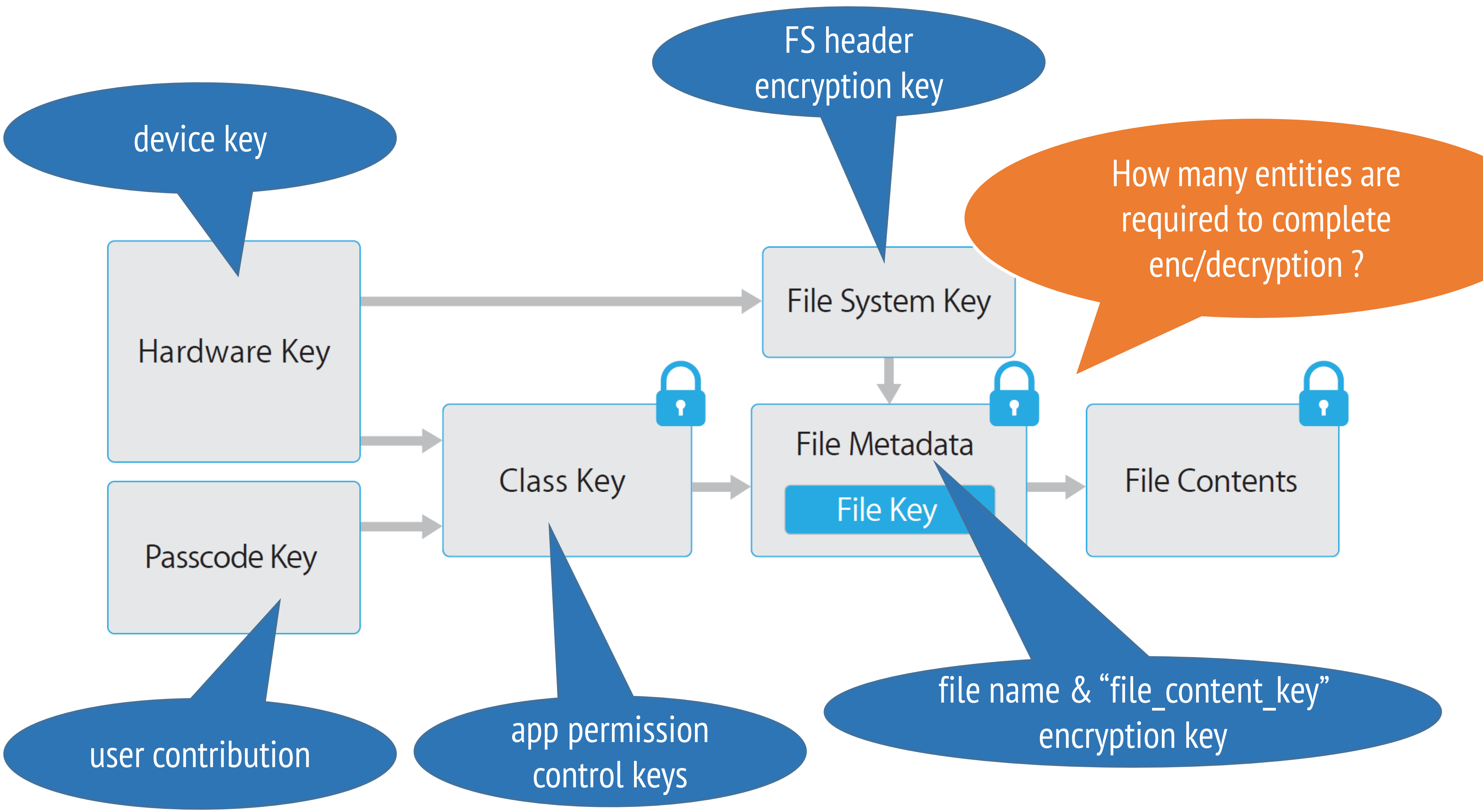
Real implementations

IOS file encrypt



https://www.apple.com/business/docs/iOS_Security_Guide.pdf

IOS file encrypt

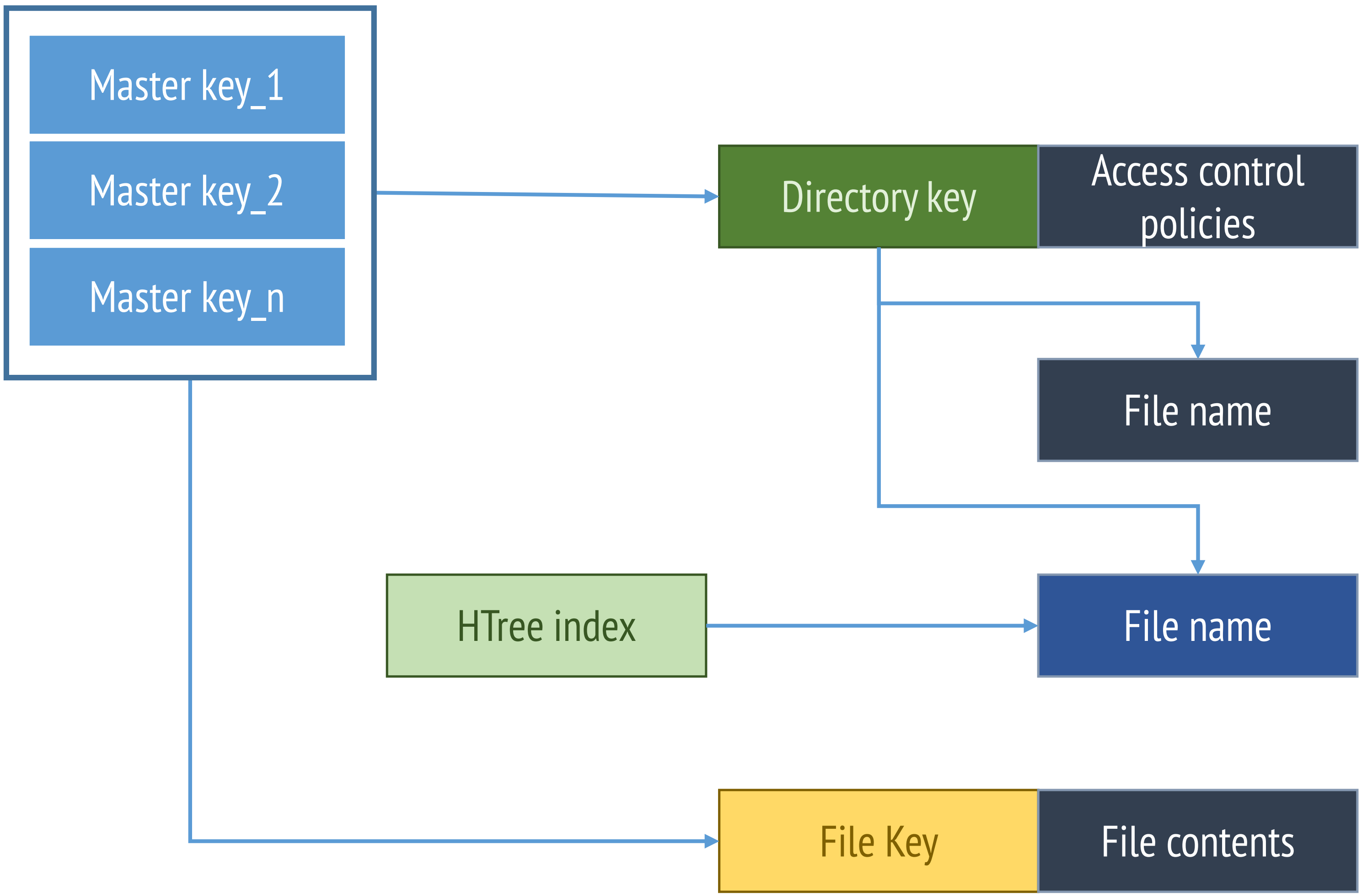


Android 7.0 (with ext4 & dm-crypt)

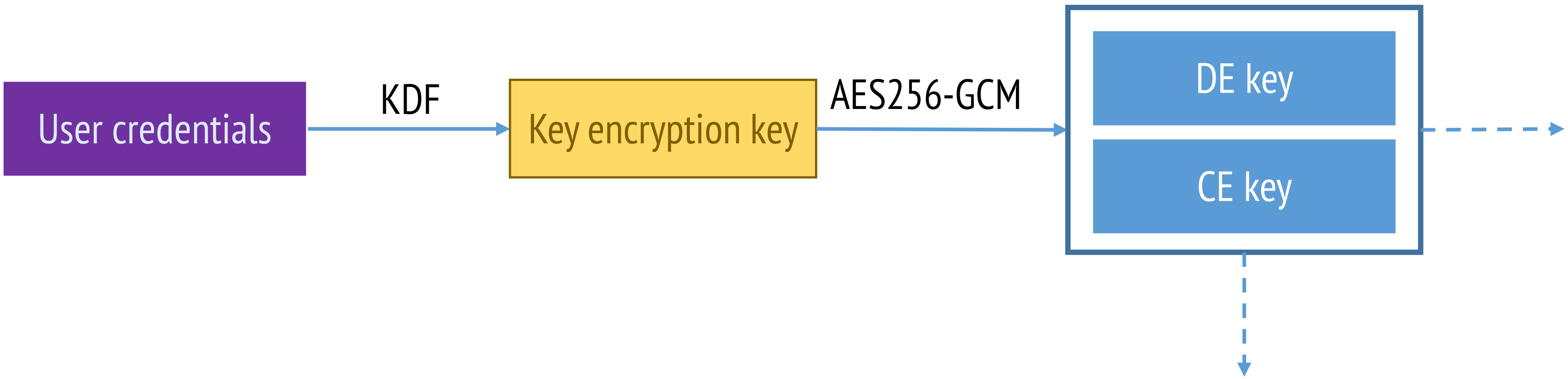
Challenges

- Follow the file structure (What does “directory” mean for ext4)
- What access control to allow if no key? (i.e. What is fail to safe?)
- How to do indexing/search?
- What can we protection can we afford? (i.e. Can we provide authentication?)

Android 7.0 (with ext4 & dm-crypt)



Android 7.0 (with ext4 & dm-crypt)



KEK is held in TEE.

Releasing KEK requires:

1. Stretched Credential: The users' authentication credentials
2. Auth Token: A cryptographically authenticated token generated by gatekeeper.
3. "secdiscardable hash": A hash of a random 16KB file that is stored for each user.

http://www.delaat.net/rp/2016-2017/p45/report.pdf

Advantages/disadvantages of file based encryption

File based encryption

- Complex design: generally many keys are used
- Does not protect metadata as well as full disk encryption
- If a key is compromised attacker gets limited access.
- More flexible

Conclusions

- Encryption provides confidentiality to data

- Full disk encryption has a simpler structure
- Full disk encryption hides metadata
- Full disk encryption usually uses one key per disk

- File based encryption has a complex structure
- File based encryption uses many keys thus is more resilient to key compromise
- File based encryption does not hide metadata as well as FDE