

Zoom Announcements

- Please keep your video camera off & microphone muted during lecture
- If you would like to ask a question:
 - Click on “chat,” then type your question. I will pause at regular intervals and answer questions posed within the chat room.
 - Click on “participants,” then use the hand icon to raise your hand. I will call on you and ask you to unmute yourself.
 - If you are calling via phone only, then please wait until I solicit questions before unmuting yourself.

Course Announcements

- Read Piazza note 227 for all updated course policies
- Homework 6 is due tomorrow (3/18)
- Homework 7 will be posted today, due Wednesday 3/25
- Midterm 2 will be converted into a project, more details posted later

Lecture 13: Protecting Data at Rest

Operating System	Product	On by default since...
Windows	Microsoft Bitlocker	Windows 8
Mac OS X	Apple FileVault	OS X Yosemite (10.10)
Linux	Linux Unified Key Setup (luks)	
Multi-OS 3 rd party tools	TrueCrypt, SecureDoc, ...	
iOS	[built into OS]	iOS 8
Android	[built into OS]	Android 6.0

Lecture outline

1. Threats and objectives
2. Sector-level encryption
3. Tweakable encryption
4. Deriving keys
5. File-level encryption

1. Threats and Objectives

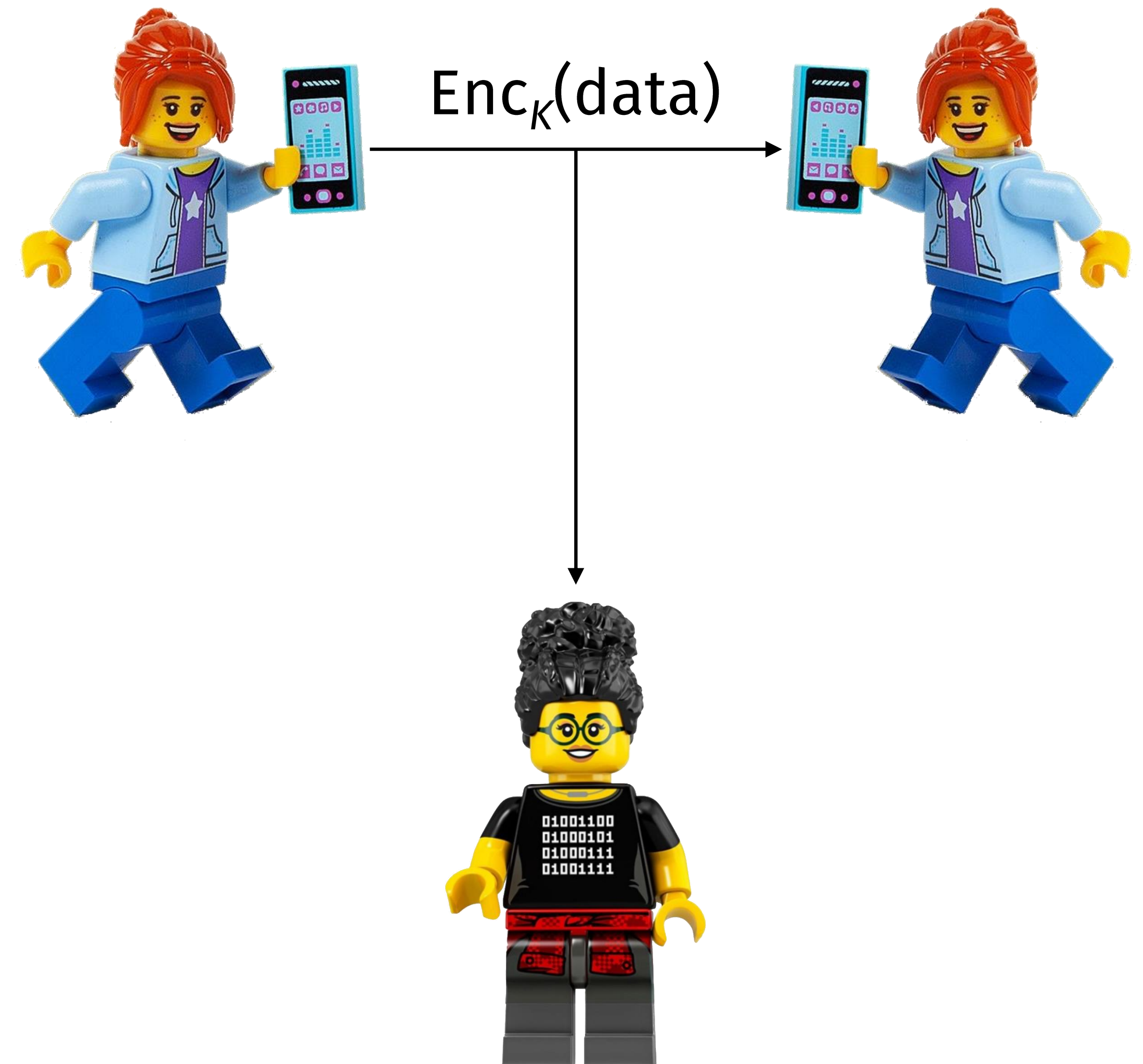
Scenario

Alice

- Knows a secret key K
- Encrypts each sector or file of her hard disk using K
- Inputs K on every device boot

Mallory

- Steals device while powered off
- Cannot exfiltrate or tamper data



Threat model

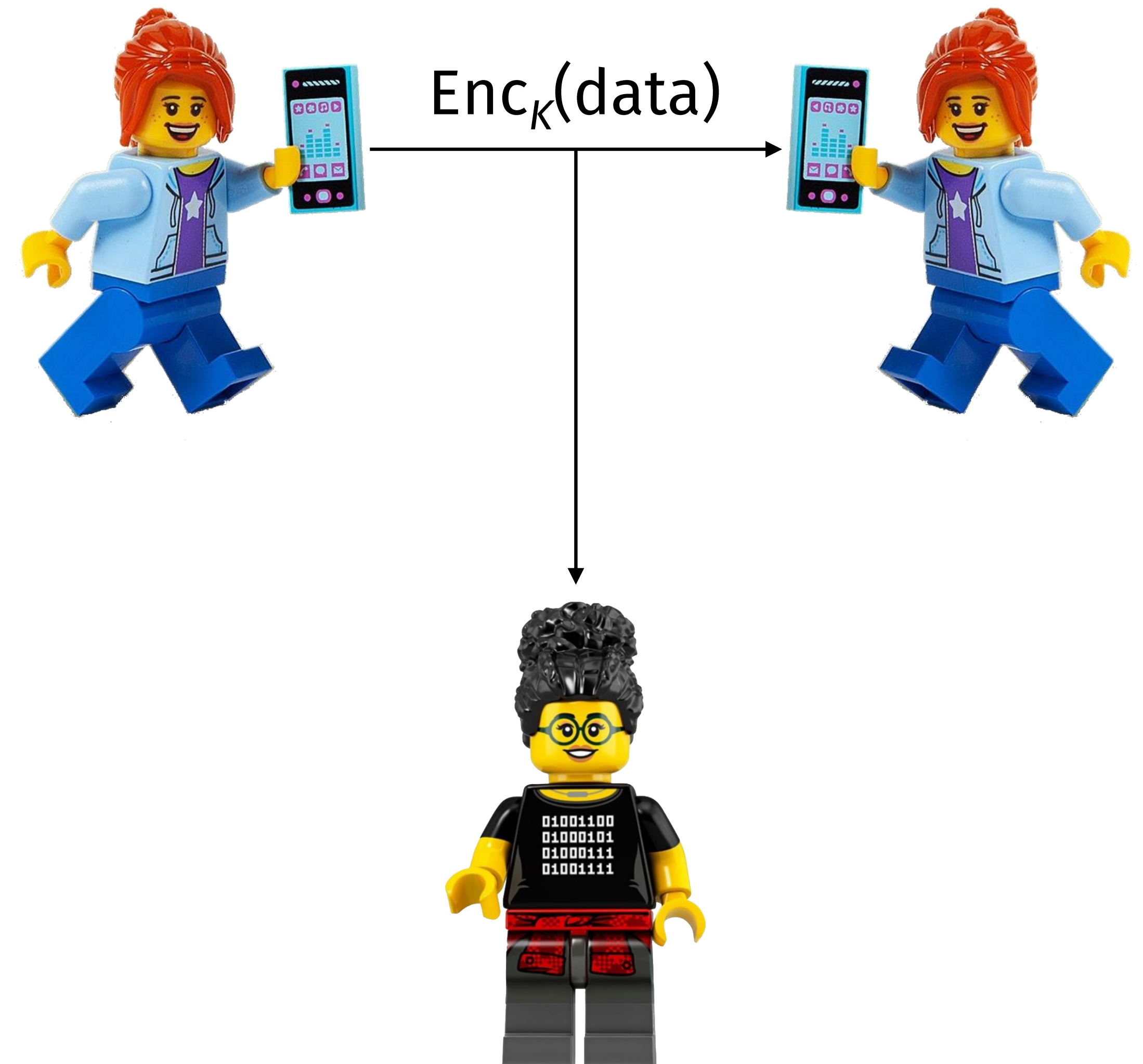
Capabilities: Mallory can

- Read encrypted contents of disk
- Encrypt new data at any location
- Modify any location of the disk
- Decrypt any location of the disk

Objectives: Mallory still cannot

- Read data (except by asking Alice)
- Tamper any location, without detection

Mallory *can* replay earlier disk contents



2. Sector-level Encryption

Overview

- The basic unit of data within a hard disk is called a "sector"
- Sectors have a fixed length specified by the disk manufacturer
 - Common options: 512, 520, 2048, 4096 bytes
 - # of sectors depends on the disk's total storage
- Objective: encrypt each sector of the disk



Components

TPM

Stores crypto keys



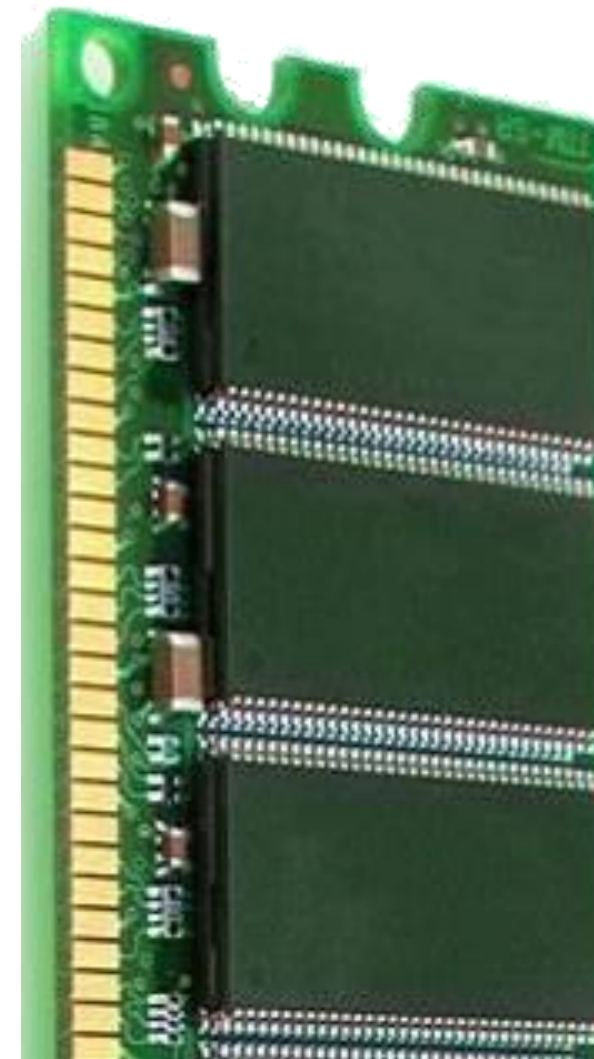
CPU

Runs crypto ops



RAM

Ephemeral storage



Hard disk

Long term storage



Workflow (first attempt)

TPM

Stores crypto keys



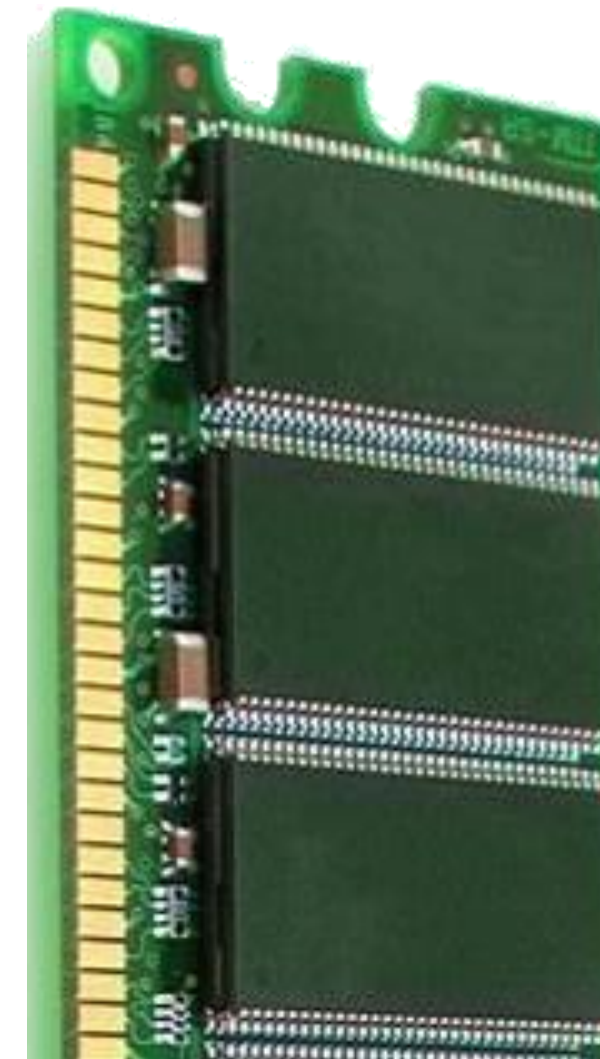
CPU

Runs crypto ops



RAM

Ephemeral storage



Hard disk

Long term storage

AES key K



$\text{Enc}_K(\text{data})$

Workflow

TPM

Stores crypto keys

CPU

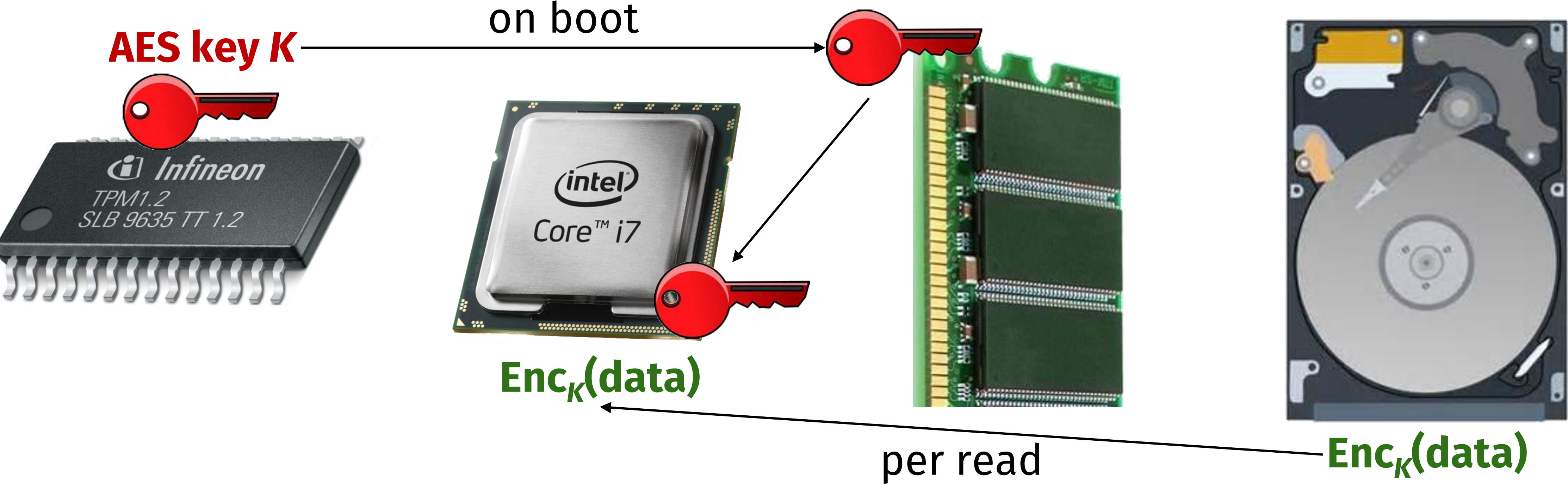
Runs crypto ops

RAM

Ephemeral storage

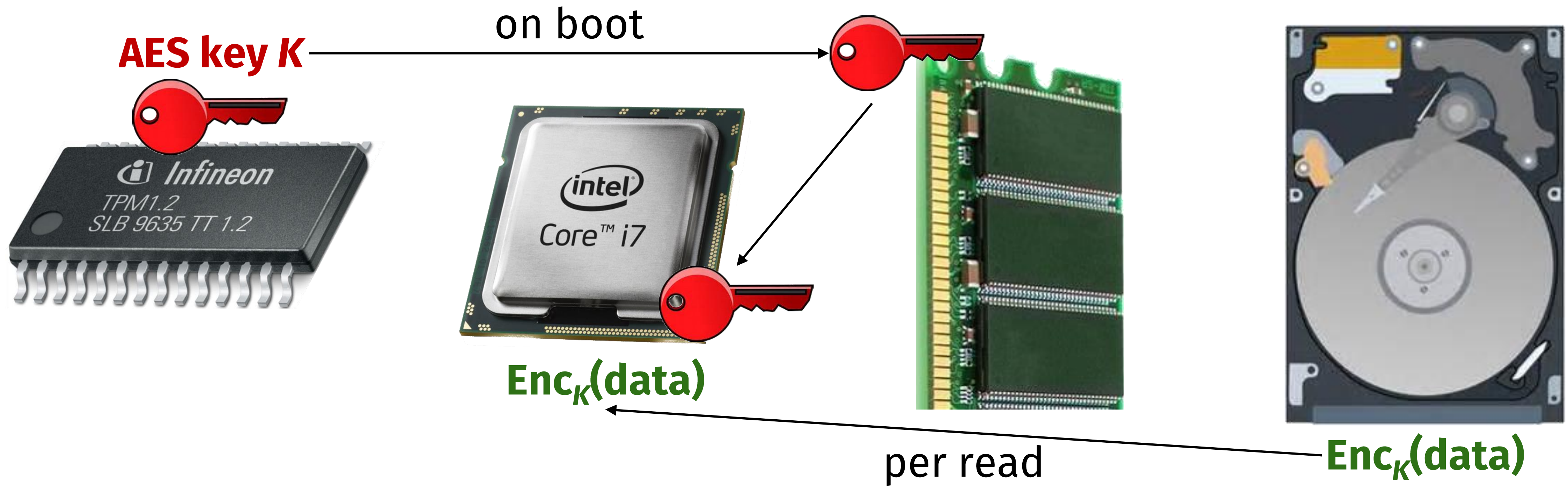
Hard disk

Long term storage



Questions

- ⇒ 1. What encryption mode of operation is safe to use?
2. How does the TPM release the key to Alice but not to Mallory?

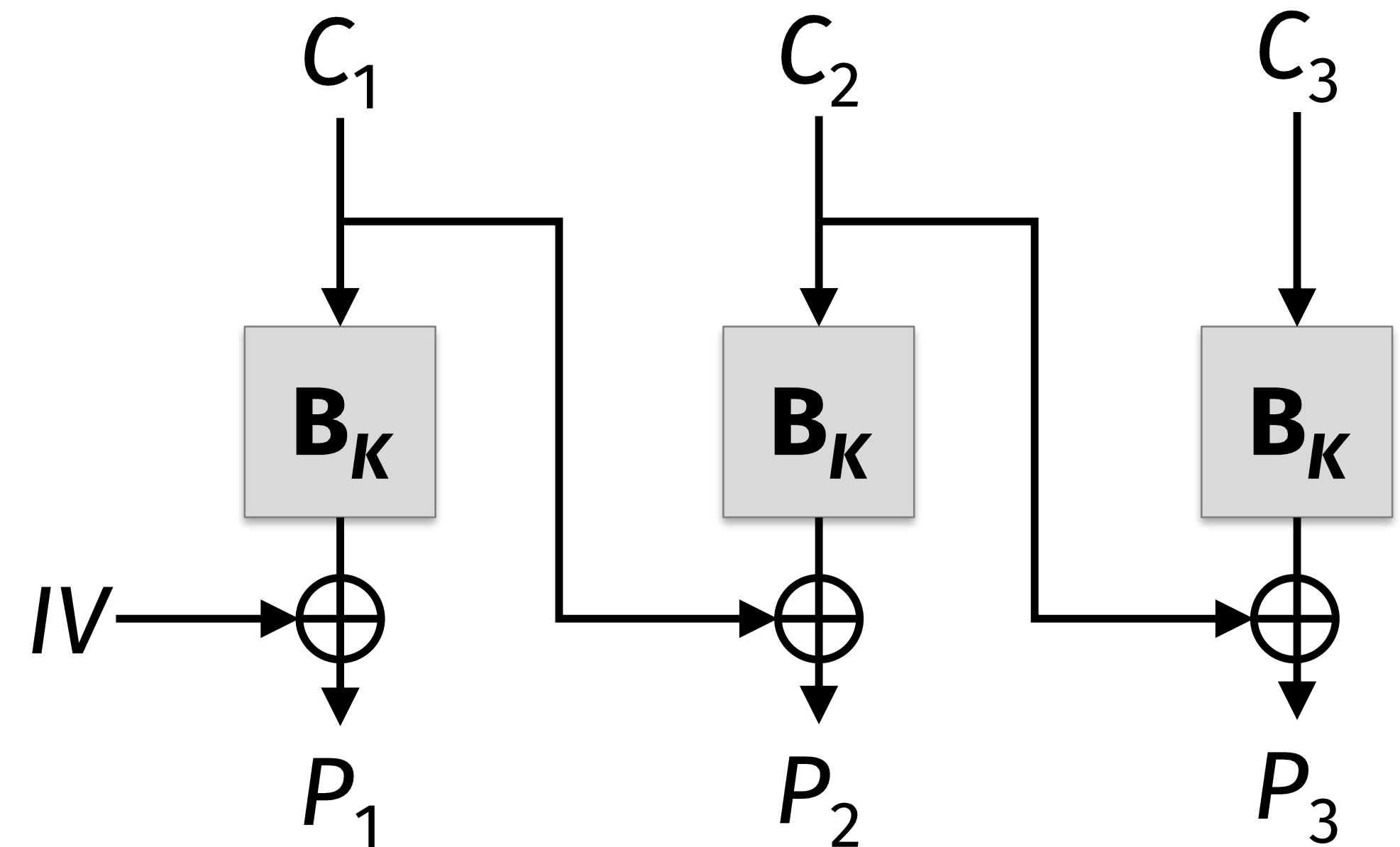


CBC mode? Lacks diffusion

If the attacker introduces a change Δ in ciphertext block i , then plaintext block i is randomized, but plaintext block $i+1$ is changed by Δ .

In other words, the attacker can flip arbitrary bits in one block at the cost of randomizing the previous block.

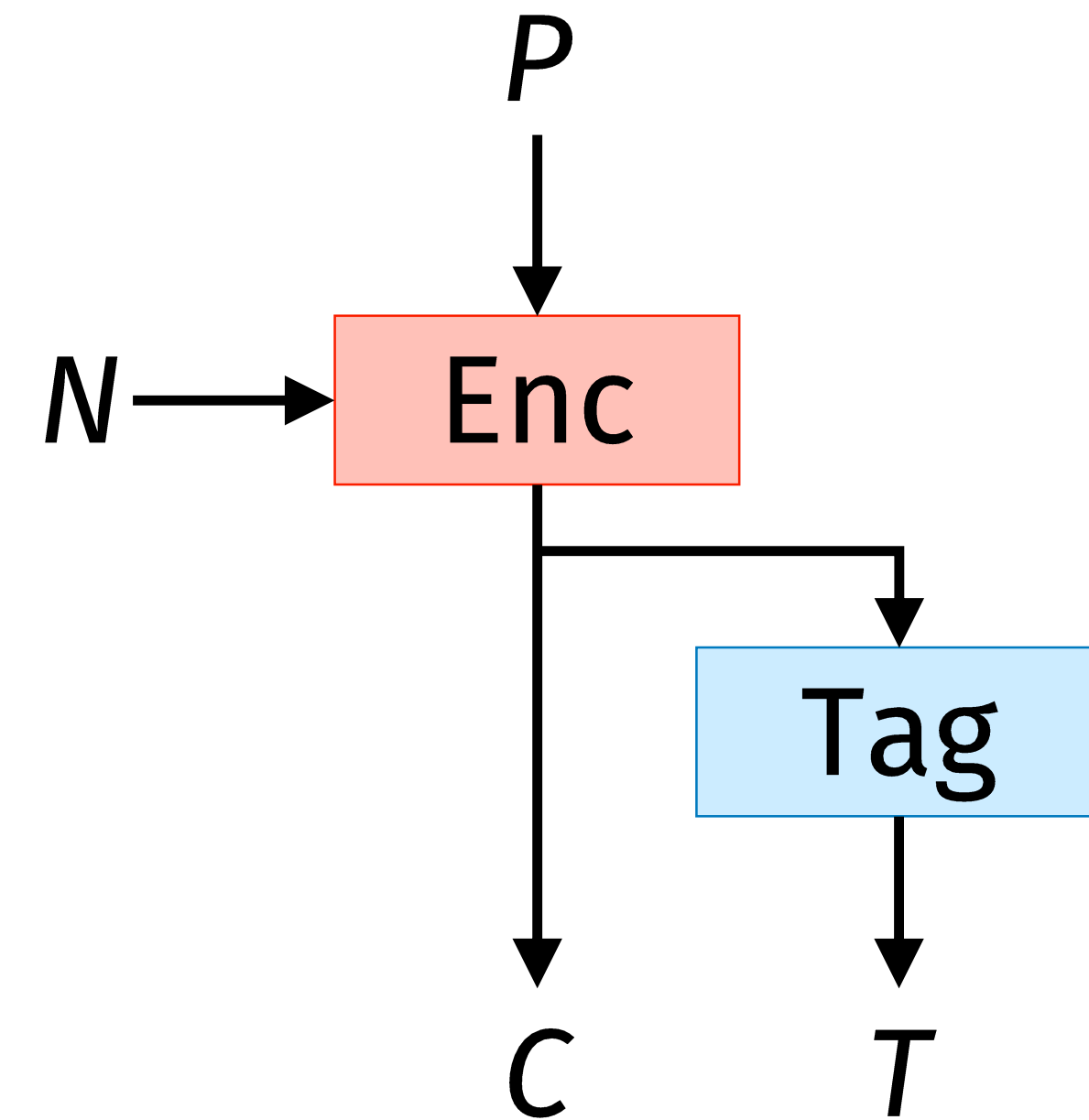
-Niels Ferguson, Microsoft, 2006



Authenticated Encryption?

Problem: $|C| + |T| + |N| > |P|$

So, authenticated encryption of 1 sector requires 2 sectors to store



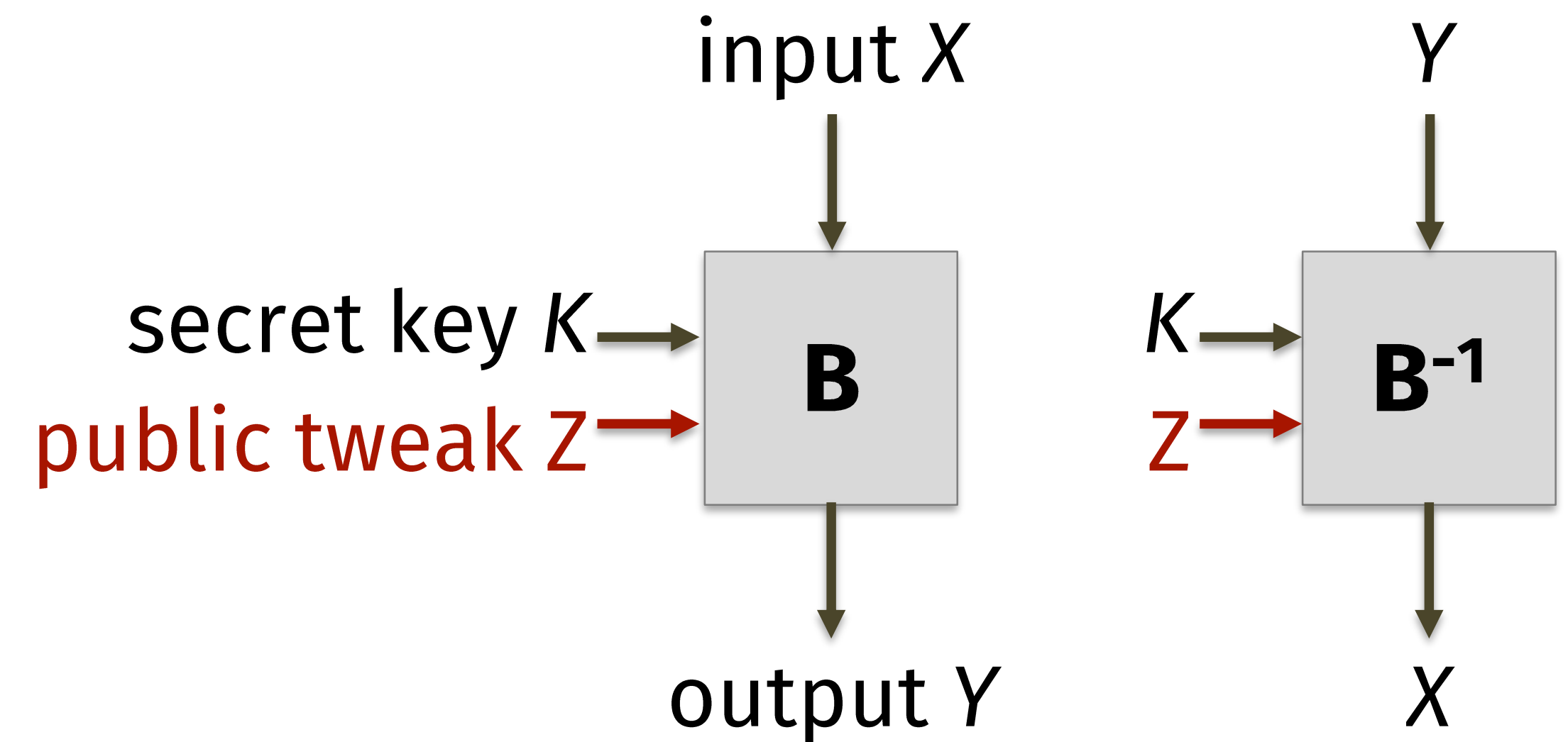
What we need for sector-level encryption

- No MAC: cannot afford the space needed to provide strong authenticity
- Still want to mitigate tampering attacks
 - Within 1 sector: want encryption mode with some resistance to tampering
 - Between sectors: encryption needs to work “independently” for each sector, so Mallory cannot move ciphertexts between sectors
- Non-goal: replay attack restoring a single sector to an earlier version

3. Tweakable Encryption

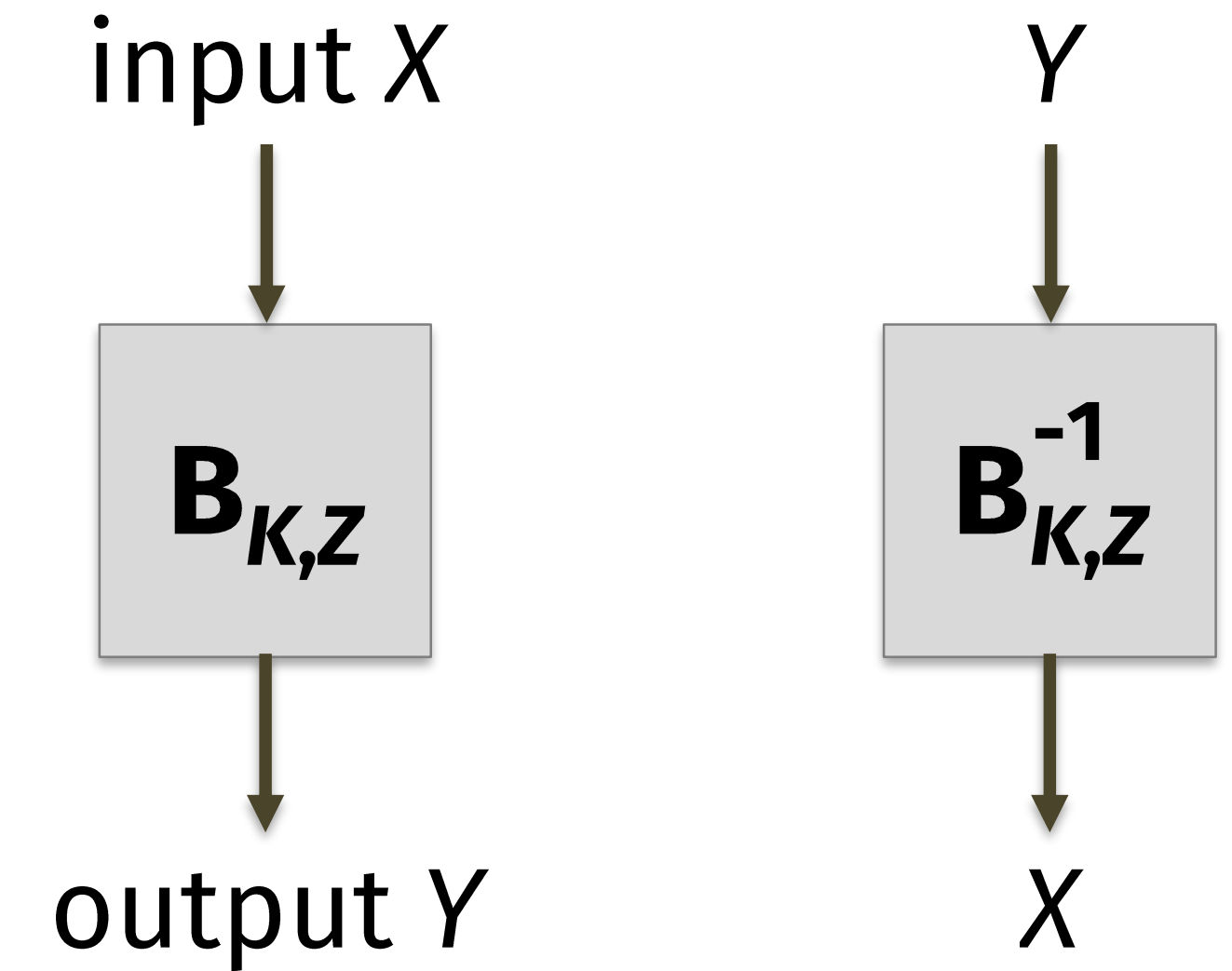
Design objectives

- Encryption builds upon block ciphers in three distinct ways
 - Add nonces for variety
 - Safe to use many times with 1 key
 - Support variable length messages
- Tweakable block ciphers incorporate the first two goals
 - Variety: changing the tweak gives an "independent" block cipher
 - Agility: changing the tweak is much faster than changing key



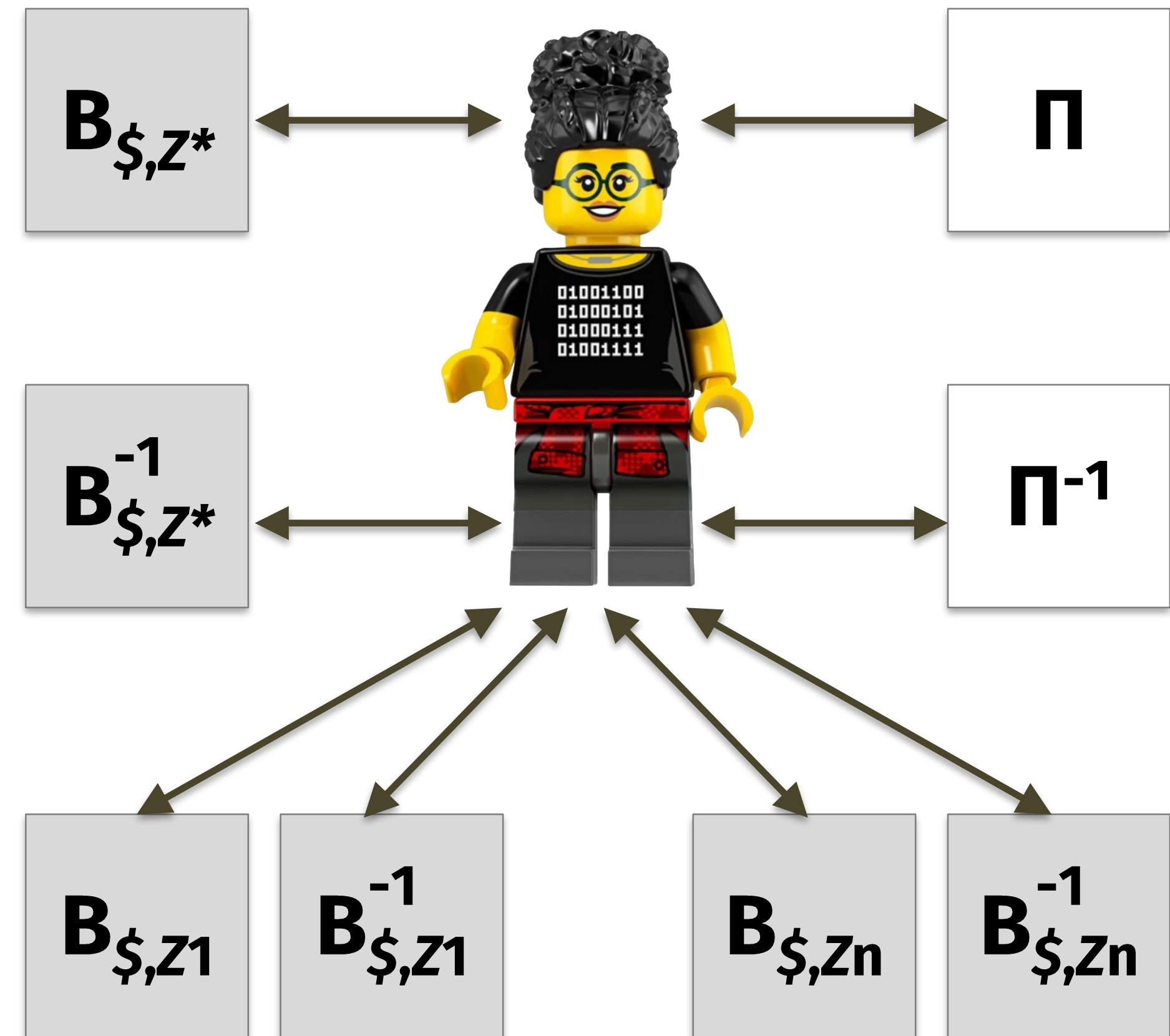
History

- Initially proposed by a 1st round AES competition candidate called “Hasty Pudding Cipher”
- Codified in 2002 by Liskov, Rivest, and Wagner



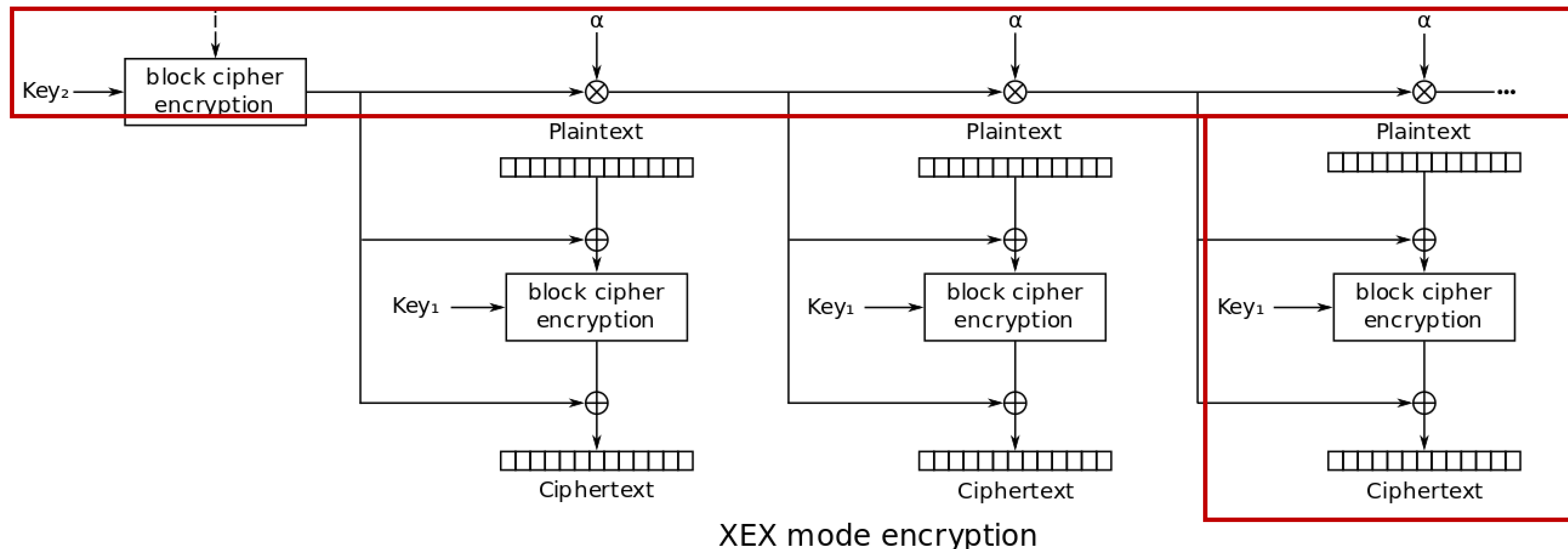
Security

- A tweakable cipher B_{K,Z^*} is strongly pseudorandom...
- Even if Mallory also has access to $B_{K,Z}$ for Z of her choice
- Note: changing the key in a normal block cipher typically does not give this property



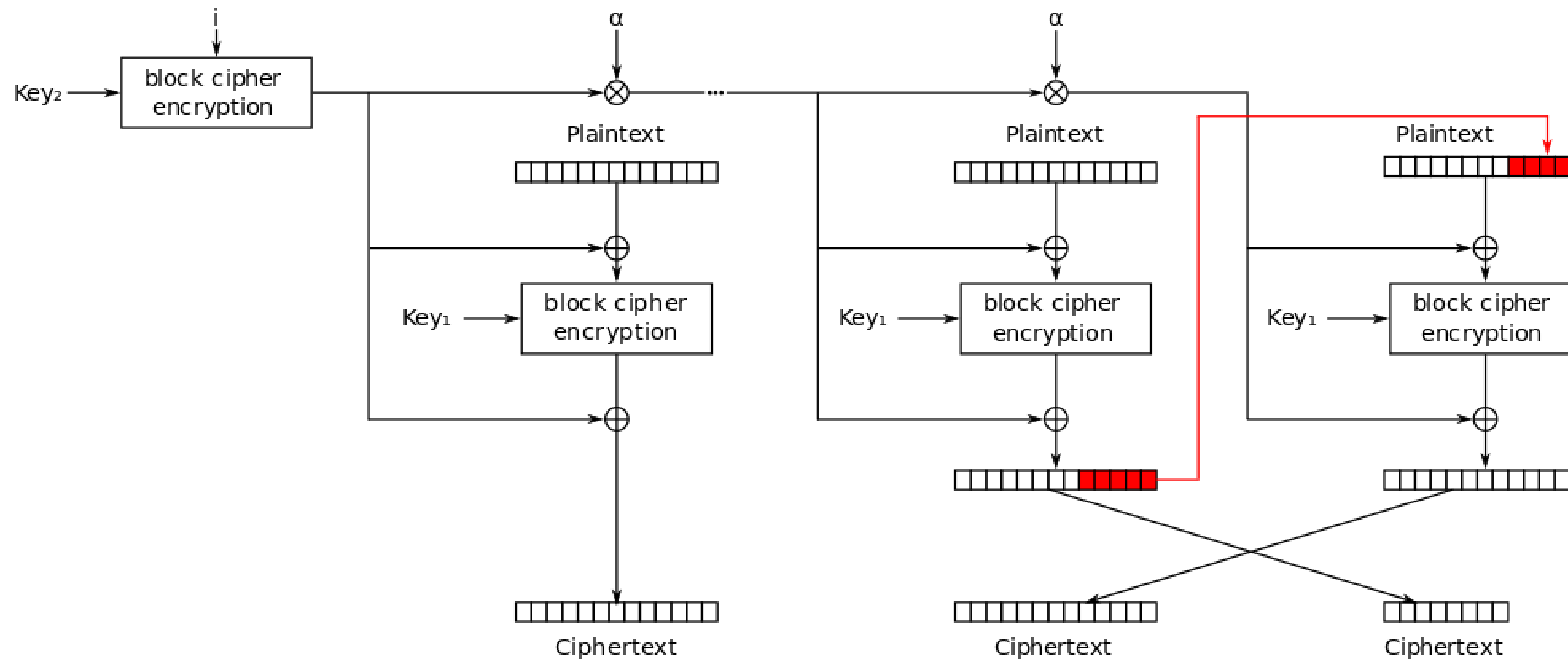
XEX mode

- Encryption mode of operation designed in 2004 by Phillip Rogaway
- Tweakable block construction has an Even-Mansour style (cf. Lecture 2)
- Each block's tweak is a function of the sector number i and secret Key_2



XTS mode

- XTS = XEX with ciphertext stealing, if sector length is not a multiple of 16
- AES-XTS is the norm for protecting disks (don't use it for anything else)

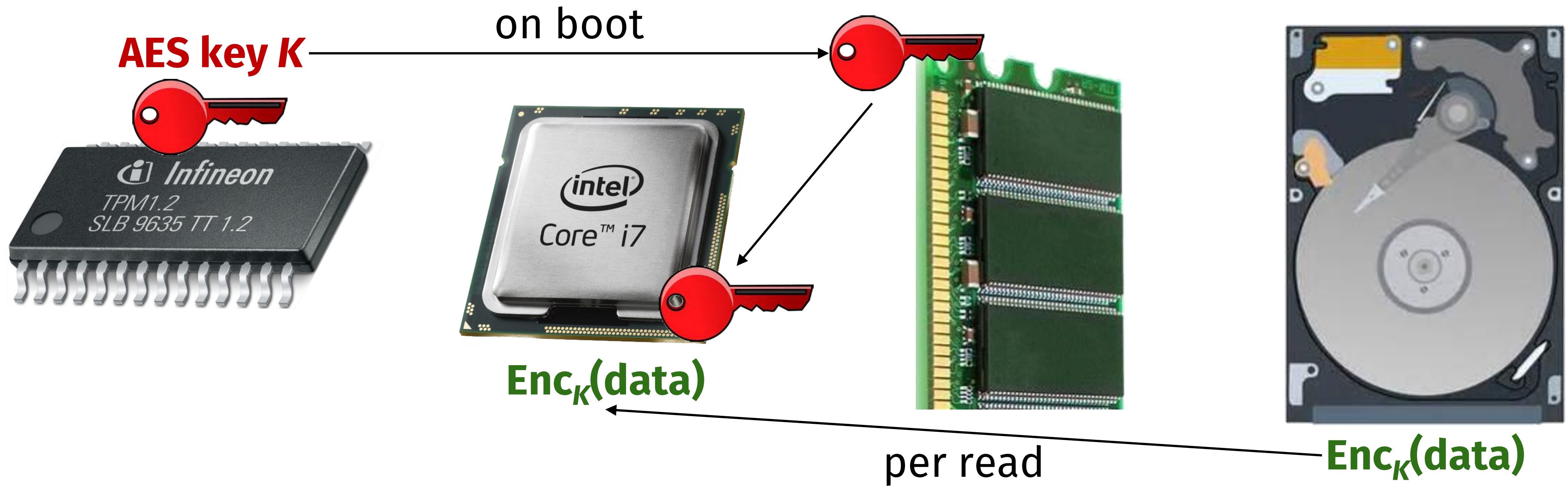


XEX with tweak and ciphertext stealing (XTS) mode encryption

Questions

1. What encryption mode of operation is safe to use?

➔ 2. How does the TPM release the key to Alice but not to Mallory?



4. Deriving Keys

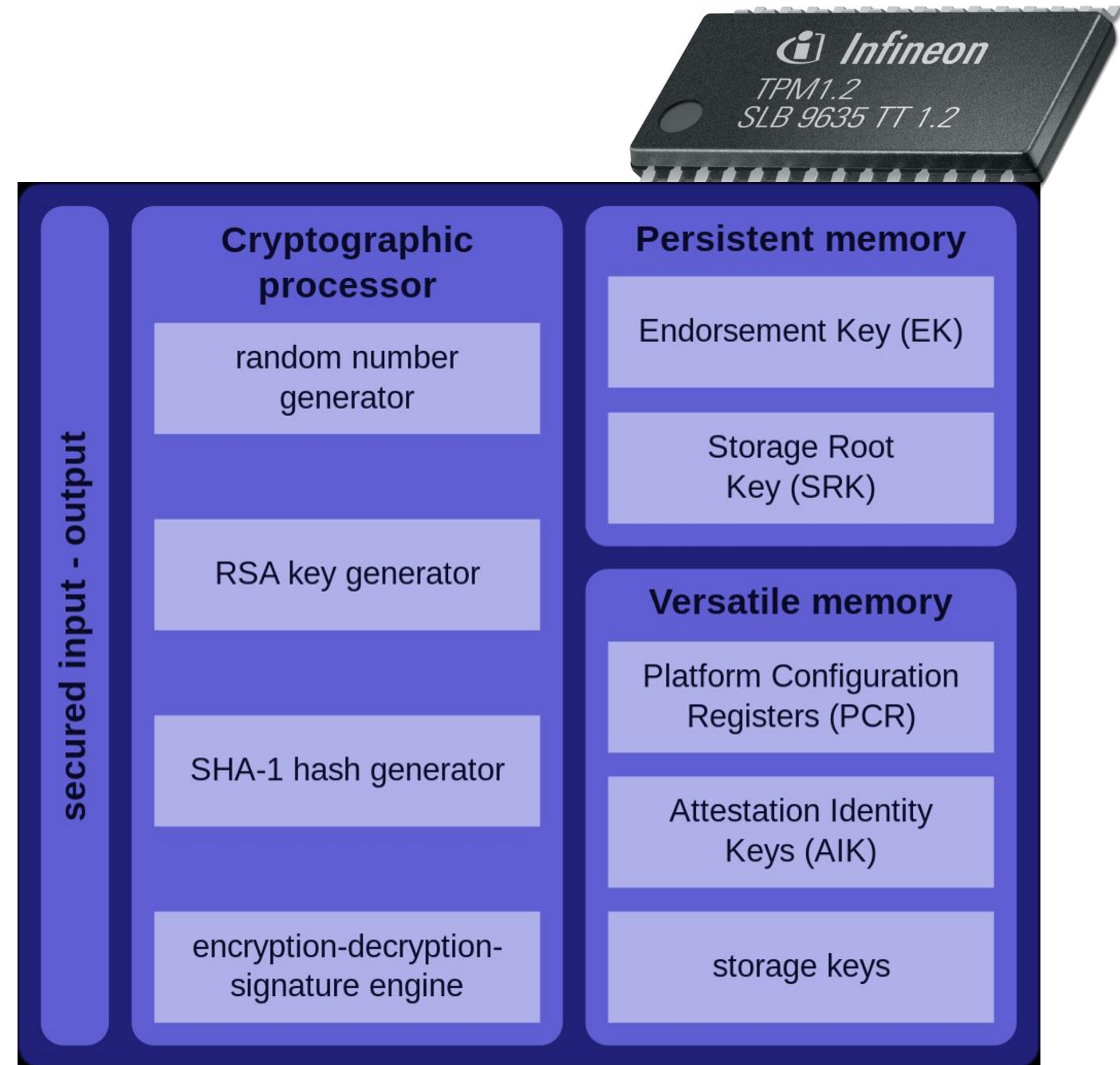
Kerckhoffs' principles to protect communication

1. The system must be practically, if not mathematically, *indecipherable*
2. It should *not require secrecy*, and it should not be a problem if it falls into enemy hands
3. It must be possible to communicate and ***remember the key*** without using written notes, and correspondents must be able to ***change or modify it at will***
- ...
6. Lastly, given the circumstances in which it is to be used, the system must be easy to use and should *not be stressful to use* or require its users to know and comply with a long list of rules

Trusted Platform Module

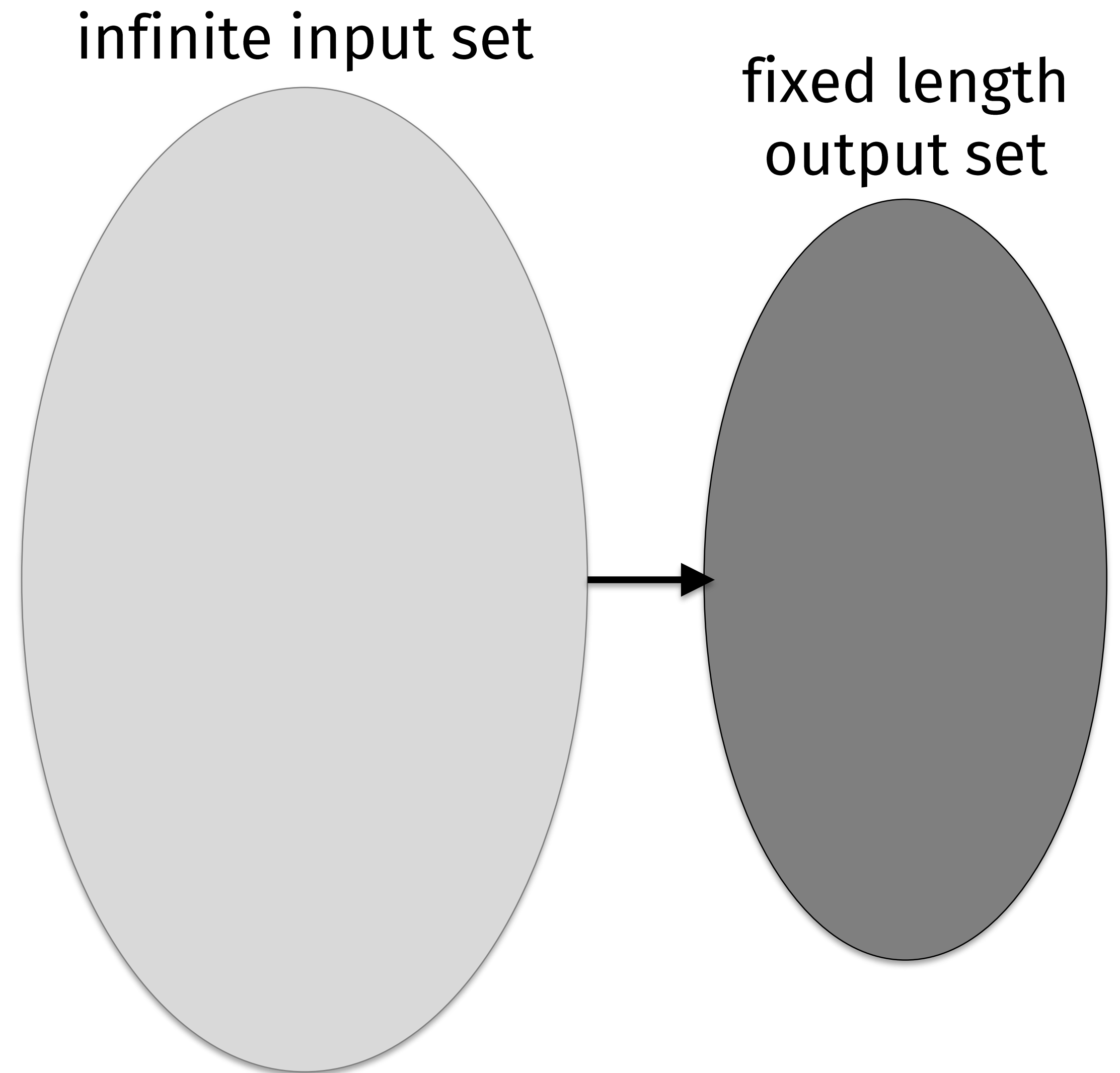
TPM generates key at boot from:

- Alice's password (using PBKDF2)
- Machine state (prevents booting into malicious OS that steals data)



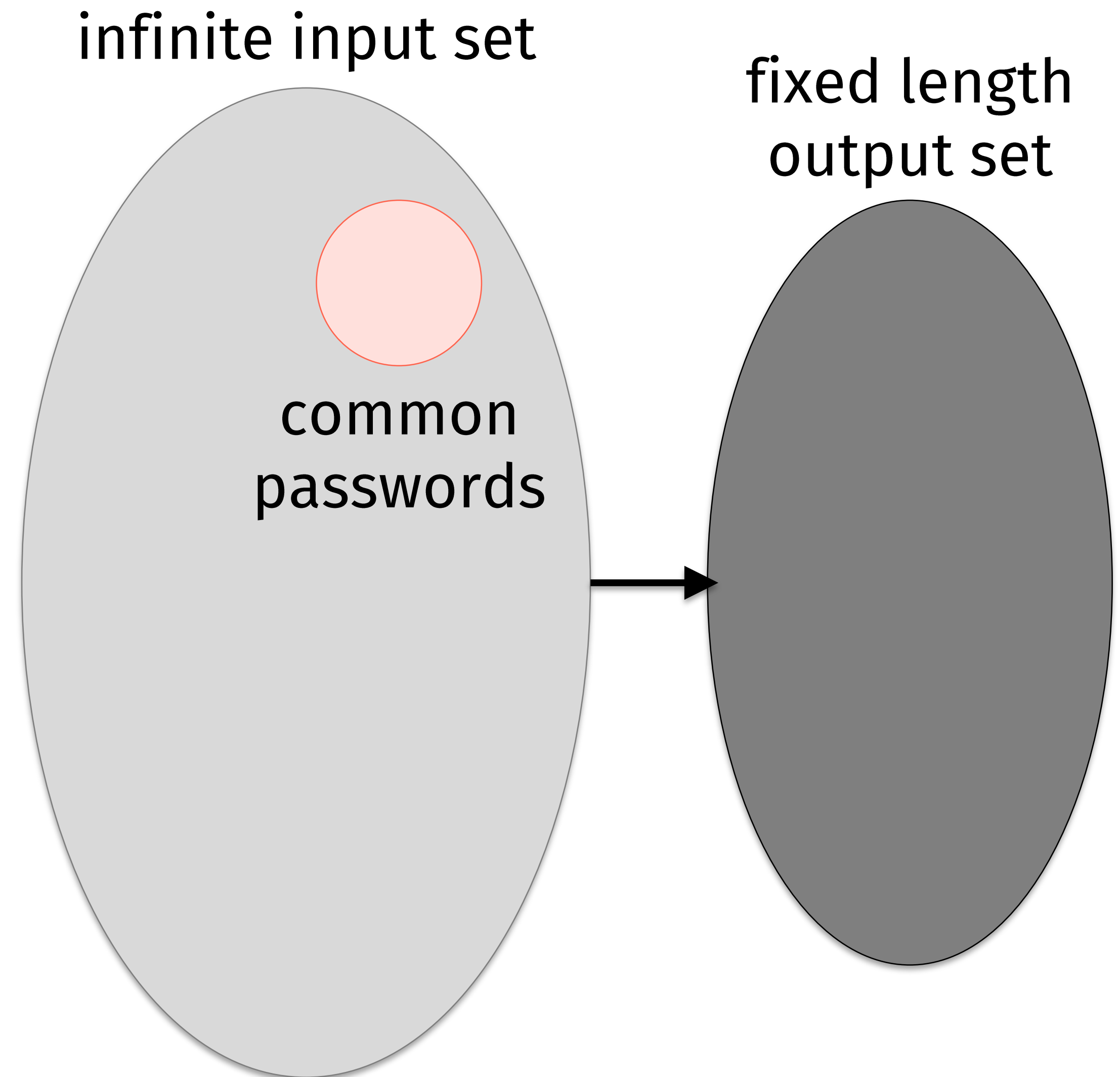
Password hashing

- Idea: generate key $K = H(W)$ by hashing Alice's password W
- Why use a hash function H ?
 - Cannot collide! People who chose unique passwords will also have a unique derived key K
 - Cannot invert! From key K , cannot find password W



Password hashing

- Problem: preimage resistance requires a uniformly-chosen W
- But people are not creative in choosing passwords
 - They follow Zipf's law:
$$\Pr[k^{\text{th}} \text{ most common password}] \propto \frac{1}{k}$$
- Mallory can brute force them!
- Countermeasure: if # of possible passwords is small, then we need a *slow* hash function



Password-based key derivation

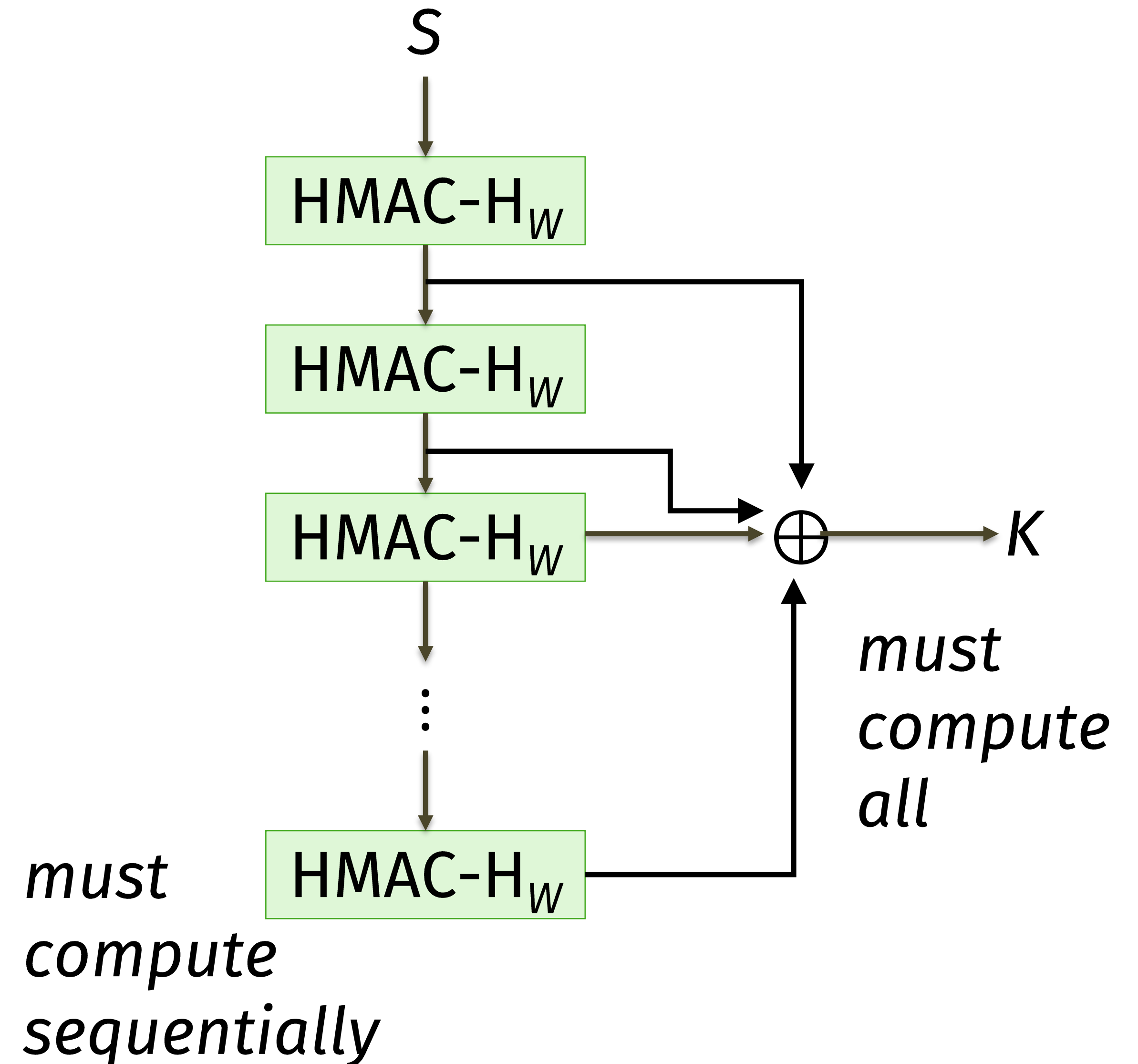
- Approach: derive key K from a moderately-strong password W
 - Best option for non-interactive login because the device never stores secrets
- Need a strong enough password to stop Mallory from brute-forcing K given enough CPU + RAM and the hashed password file in `/etc/passwd`
- *Offline dictionary attack*: brute-force on Mallory's machine beforehand
- Crypto primitives
 - PBKDF2: NIST standard, requires substantial CPU time
 - Newer password hash functions like scrypt, bcrypt, and argon2 also require significant RAM to compute, in an attempt to reduce parallelization

PBKDF2

Inputs

- W : Alice's password
- C : iteration count
- S : salt, aka nonce
- L : output length, typically equals hash function H 's output length

Output: key K



Key wrapping

- What if several keys should grant access to the disk?
 - Different accounts on one machine
 - Recovery key stored in a separate, safe place
- Don't want to encrypt the entire drive several times!

$\text{Enc}_{K_1}(\text{sector})$

$\text{Enc}_{K_2}(\text{sector})$

$\text{Enc}_{K_3}(\text{sector})$

Key wrapping

- What if several keys should grant access to the disk?
 - Different accounts on one machine
 - Recovery key stored in a separate, safe place
- Don't want to encrypt the entire drive several times!
- Key wrapping = protect one key under another
 - Ordinary encryption does not suffice to protect keys
 - Need Deterministic Authenticated Encryption (use SIV mode)

$\text{Wrap}_{K_1}(ek)$

$\text{Wrap}_{K_2}(ek)$

$\text{Wrap}_{K_3}(ek)$

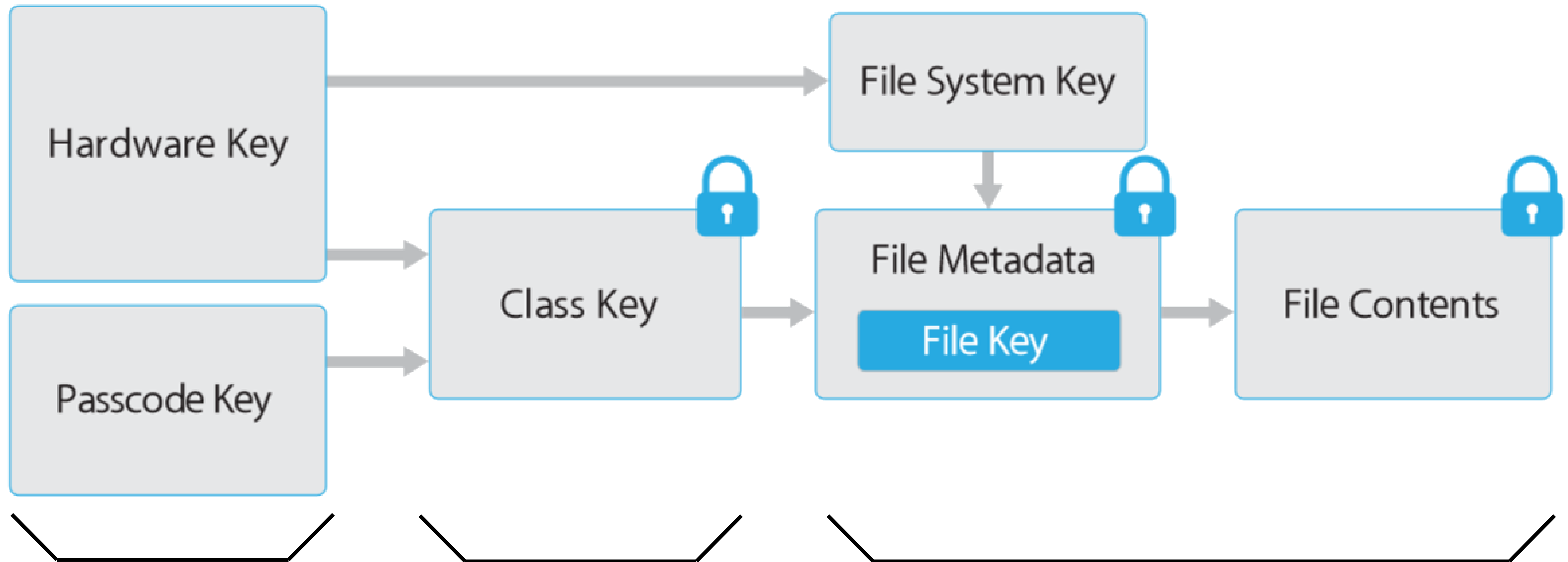
$\text{Enc}_{ek}(\text{sector})$

5. File-level Encryption

Case study: Apple's iOS

- Encrypt data at the filesystem level
- Key wrap so that data can only be decrypted in the right *context*
- Hardware provides some protection against brute-forcing passwords

Hierarchy of keys



Master key generated from things that you know, are, and have

Keys are available for a limited time, and only when needed

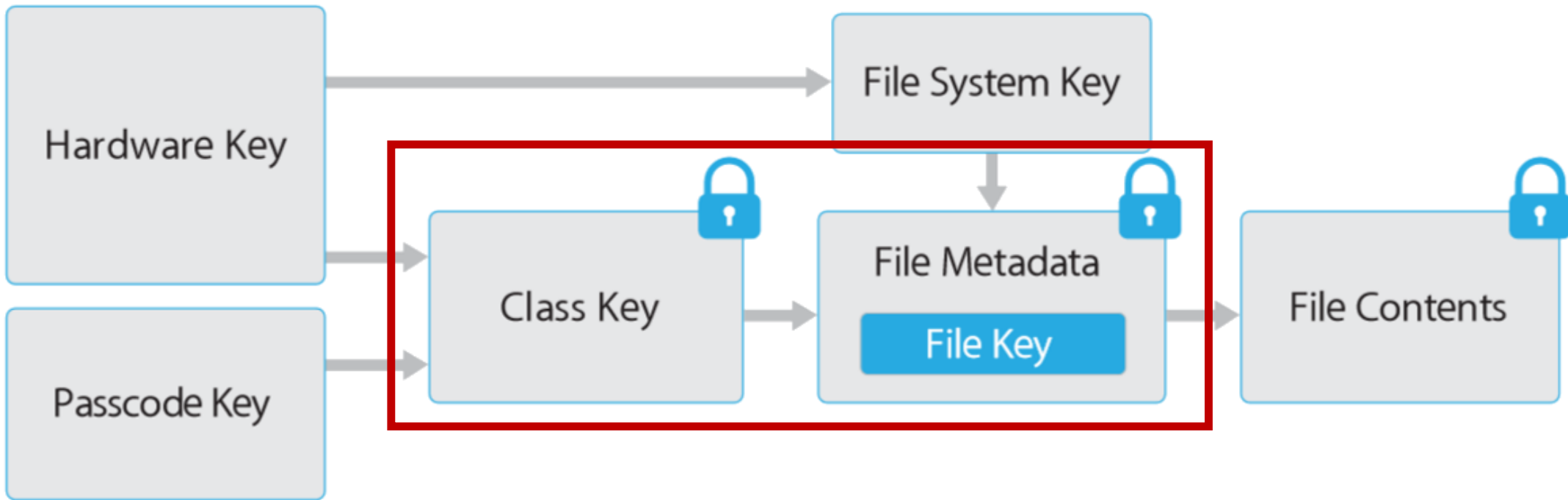
Each file is encrypted with a unique key using AES-XTS. Working within filesystem gives space to store metadata on the side.

Class keys → file keys

Per-file key is wrapped with 1 of 4 “class keys” based on availability

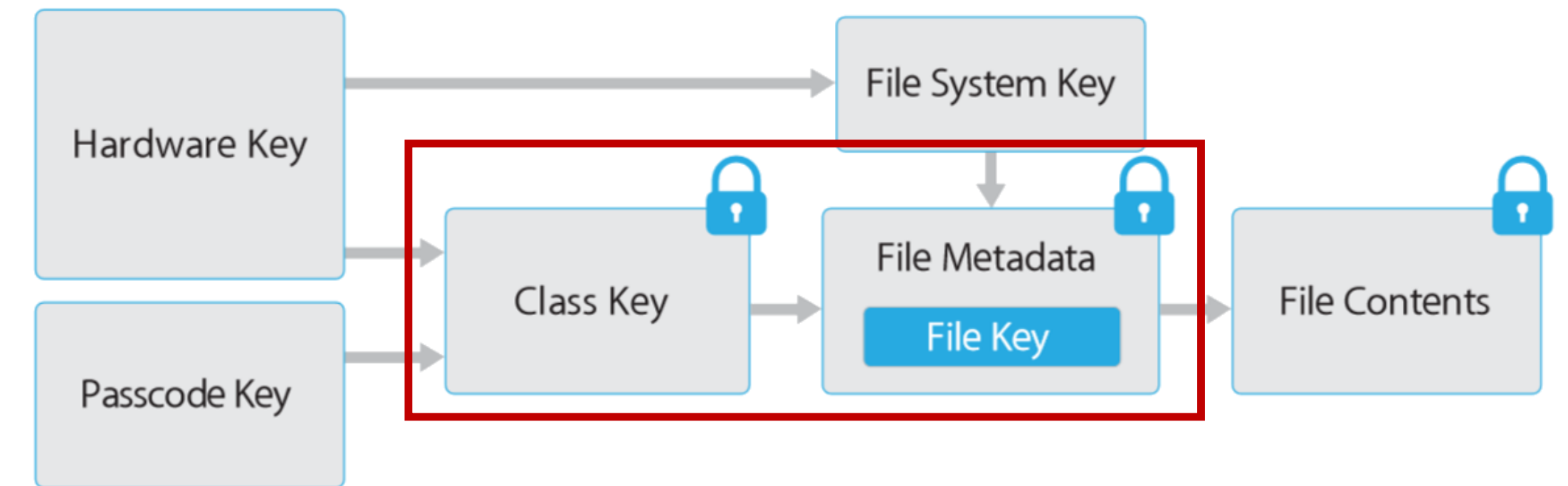
Availability	Example	Key erased if phone is...
Always	SIM PIN	Wiped
After 1st unlock	Wifi password	Shut down
When unlocked	Browser bookmarks	10s after lock (without biometric)

When locked Incoming email (works differently)

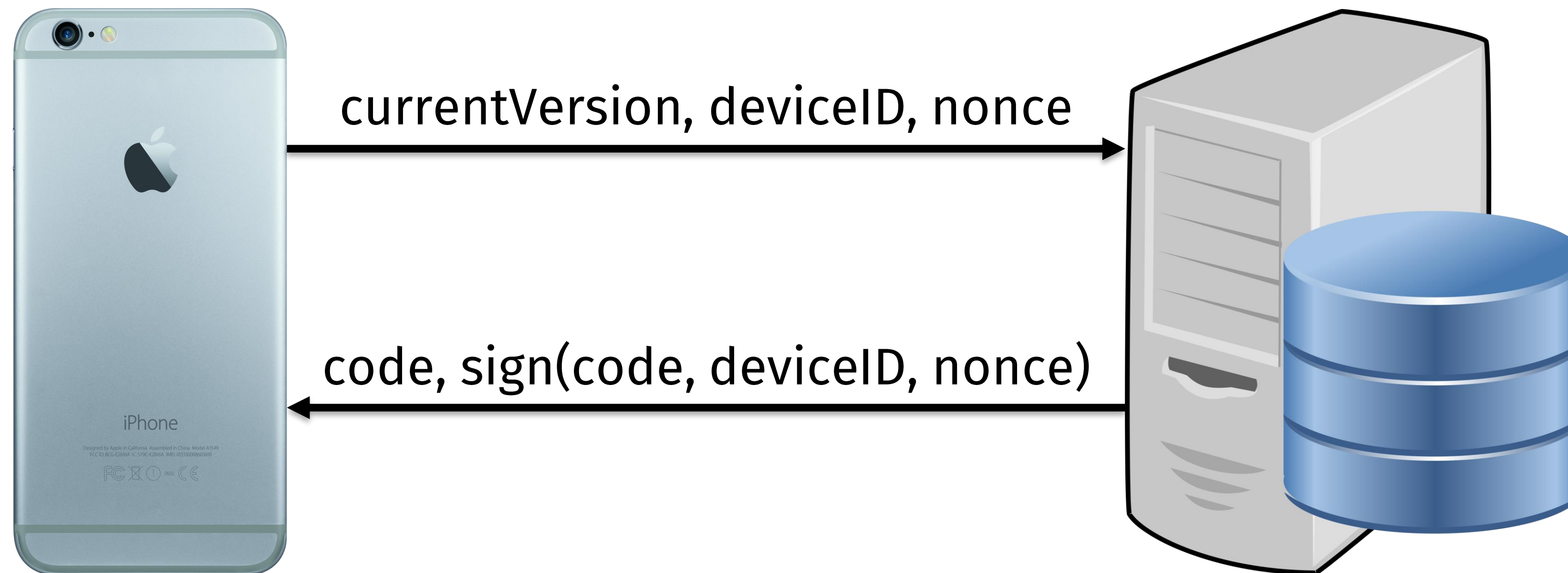


Secrets → class keys

- Rather than *storing* keys on the device, *derive* it from secrets
 - 256 bit string fused into phone
 - Alice's password
- Slow Mallory using
 - Crypto: 80ms per attempt
 - Hardware: increasing delays between attempts
 - Optionally, wipe the phone



iOS Software Updates



- Device ID *personalizes* the server's response to this particular phone
- Nonce ensures that response is *fresh*, prevents replay attacks
- Need: a *public signature scheme* that anybody can verify!