Case Studies

PROFESSIONAL EDUCATION

CSAIL
MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY

Disk Encryption: BitLocker

Nickolai Zeldovich

Associate Professor

Computer Science and Artificial Intelligence Laboratory (CSAIL)

Massachusetts Institute of Technology

PROFESSIONAL EDUCATION

CSAIL
MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY

Segment 1: Introduction

- Goals and threat model
- Approaches
- Challenges

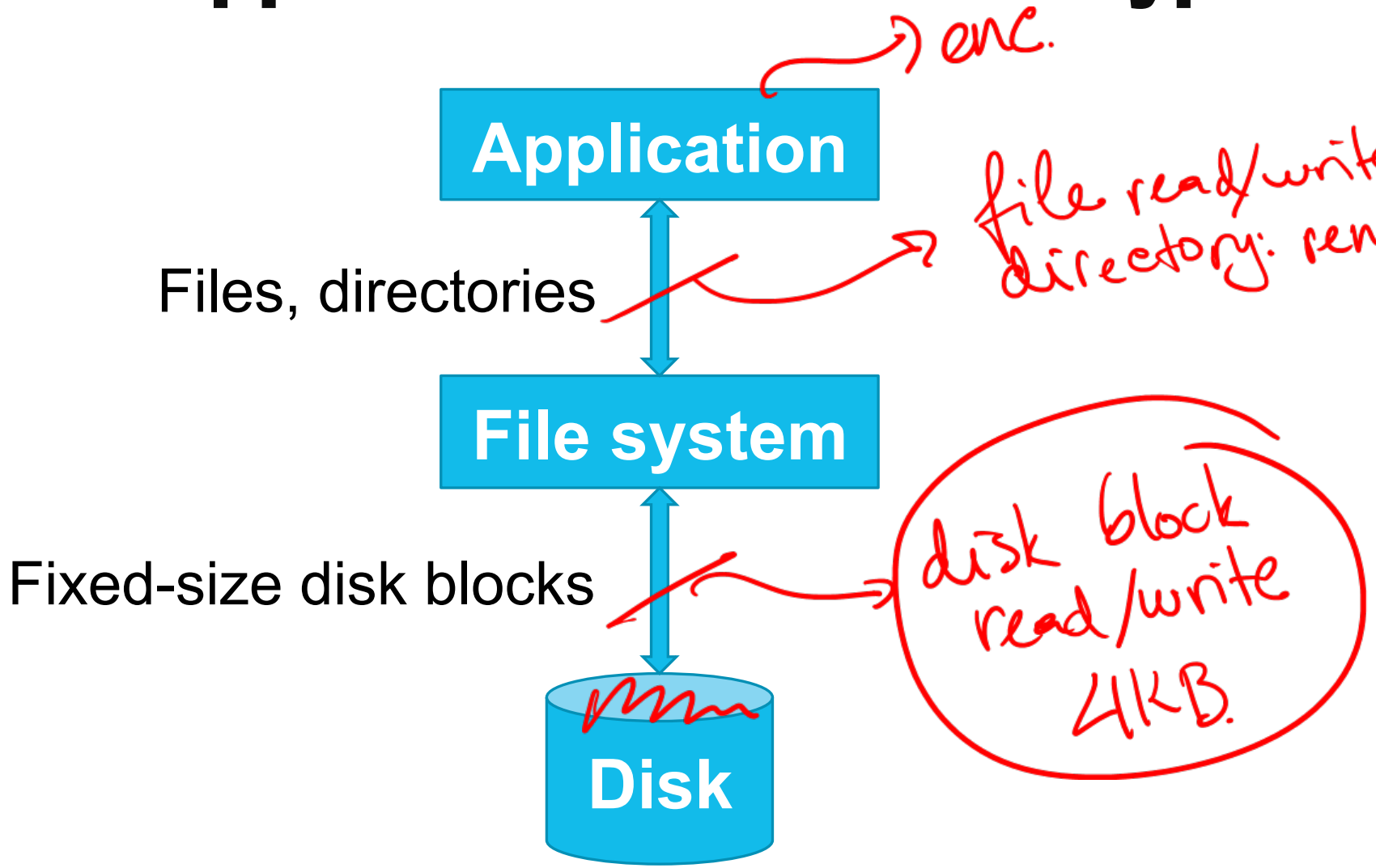
Disk encryption goals

- Scenario: adversary steals a laptop
- Primary goal: *secrecy*
  - Ensure confidential data not leaked to adversary
- Related goal: *integrity*
  - Adversary cannot tamper with data on disk
  - Will be indirectly important later on
- Others: *freshness*, *deniability*, …

Threat model

- What can the adversary do?
  - Boot up the laptop
  - Boot from a CD or USB drive, change password
  - Remove disk, plug into another computer
  - Make copies of the disk, so as to try different things
- What is the adversary *unable* to do?
  - Break cryptography, use lots of compute power
  - Break tamper-resistant hardware
  - Know the user's exact password, PIN, …

Approaches to disk encryption



Challenges in disk encryption

- Where does the key come from?
  - Password, USB device, trusted hardware (TPM), …
- How is the data encrypted on disk?
  - Using block ciphers, randomness, authentication, …

Segment 2: Keys from passwords

- The trouble with passwords: entropy
- Key-derivation functions
- Salting passwords

Turning a password into a key

- Password: variable-length string
- Key: fixed number of bits
- Naïve approach: use a hash function

key = SHA1(password)

Problem: passwords have low entropy

- Users choose predictable passwords
  - Top 5,000 passwords account for 20% of users (out of 32M)
- Our threat model: adversary can guess
  - Guess requires hashing password, seeing if decryption works
  - Decryption must be designed to be fast
  - Hash functions also designed to be fast
  - Adversary can make over 1M guesses/second

Idea 1: expensive key derivation

- Plan: make the hash function expensive

key = SHA1(SHA1(…SHA1(password)..))

- Slows down each guess by adversary
- Not a big problem for user: just one time

Problem: pre-computed tables

- Adversary can pre-compute table of hashes
  - Time-consuming, but worth it to break passwords quickly!

Password	Hash
aaa	7e240de7…
acorn	f5d193d9…
alice	522b276a…
…	…

Idea 2: salting the passwords

- Introduce random "salt" into key generation

$$key = H(password, salt)$$

- To compute key, need password *and* salt
  - Salt stored in plaintext on disk, but crucially, has high entropy
  - Adversary can get the salt value after stealing the laptop
  - However, now cannot pre-compute table of possible hashes!

Real-world key derivation functions

- PBKDF2
- bcrypt
- scrypt

Limitations of using passwords

- Susceptible to guessing, even with salting
  - Ideally, want to allow just a handful of guess attempts!
- Usability challenges
  - Disk encryption key needed as soon as computer boots up
  - User must enter password into BIOS, before OS running
  - OS may require separate authentication: password, fingerprint, ...
- Nonetheless, can be useful for recovery

Segment 3: Password alternatives

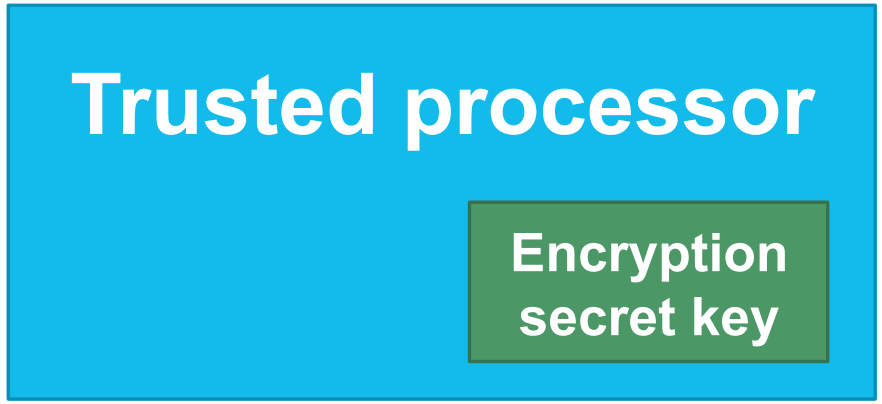
- Removable devices
- Trusted hardware
- Signed boot

Removable devices

- Idea: store key on removable device
  - USB drive, or other devices (e.g., Yubikey)
- Good: strong, hard-to-guess key
  - Can generate high-entropy keys, no need to rely on user
- Bad: user has to carry separate device
- Bad: device could well be in stolen laptop
  - Device has no idea who is using it (adversary or user)

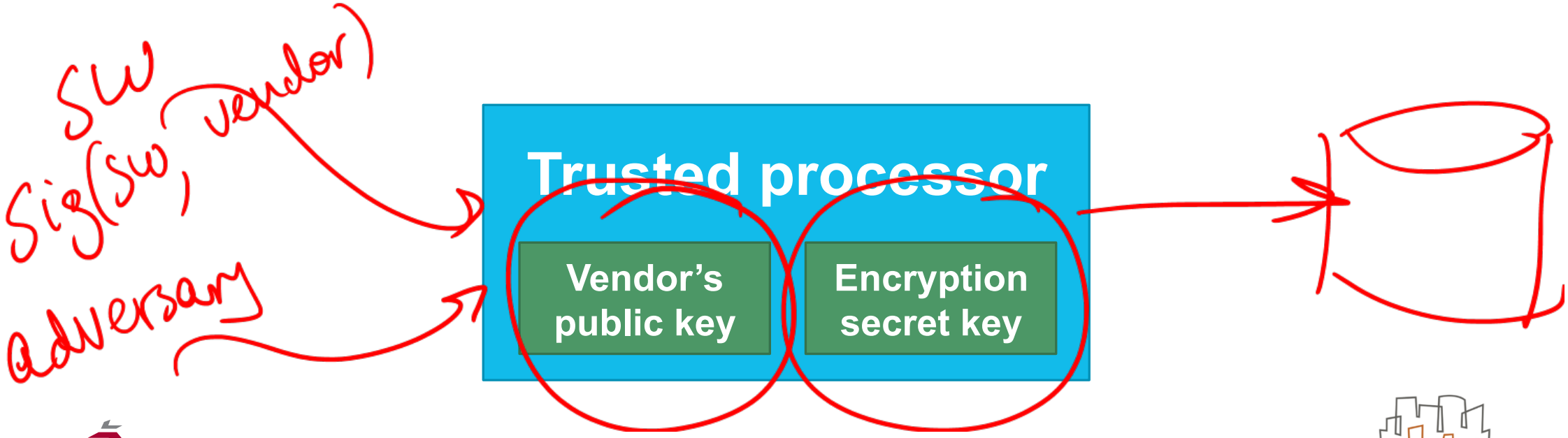
Trusted hardware

- Idea: store key in a tamper-proof processor
  - E.g., system CPU or a nearby chip
  - Ensure that physically “capping” the chip is costly
- What software is allowed to access the key?



Signed boot

- System requires all software to be signed
  - Vendor's public key hard-wired into the trusted processor
  - Vendor signs only approved software releases
  - E.g., used by some game consoles, cell phones, etc



Disk encryption with signed boot

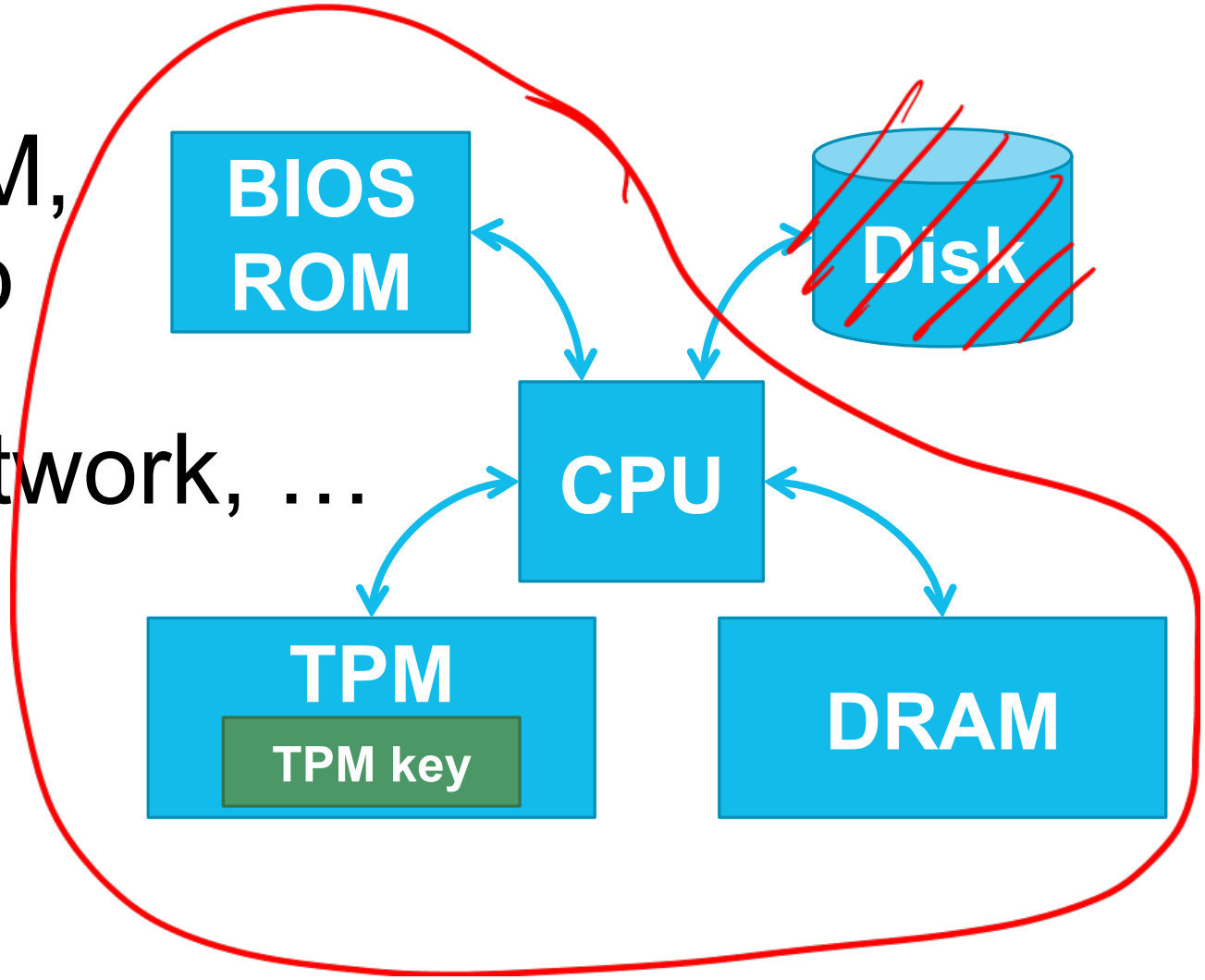
- Use key in trusted processor for disk enc.
- Signed OS implements user authentication
  - Passwords, fingerprints, face recognition, …
  - No need to generate key from authentication mechanism
  - Easy to enforce strong limits on guessing in software
- Downside: too restrictive in many cases
  - All OS code must be signed by a single vendor

Segment 4: Trusted Platform Module (TPM)

- System model
- TPM interface
- Measured boot

TPM system model

- Trusted: CPU, DRAM, BIOS, and TPM chip
- Not trusted: disk, network, …
- TPM keeps track of what software runs, selectively uses key



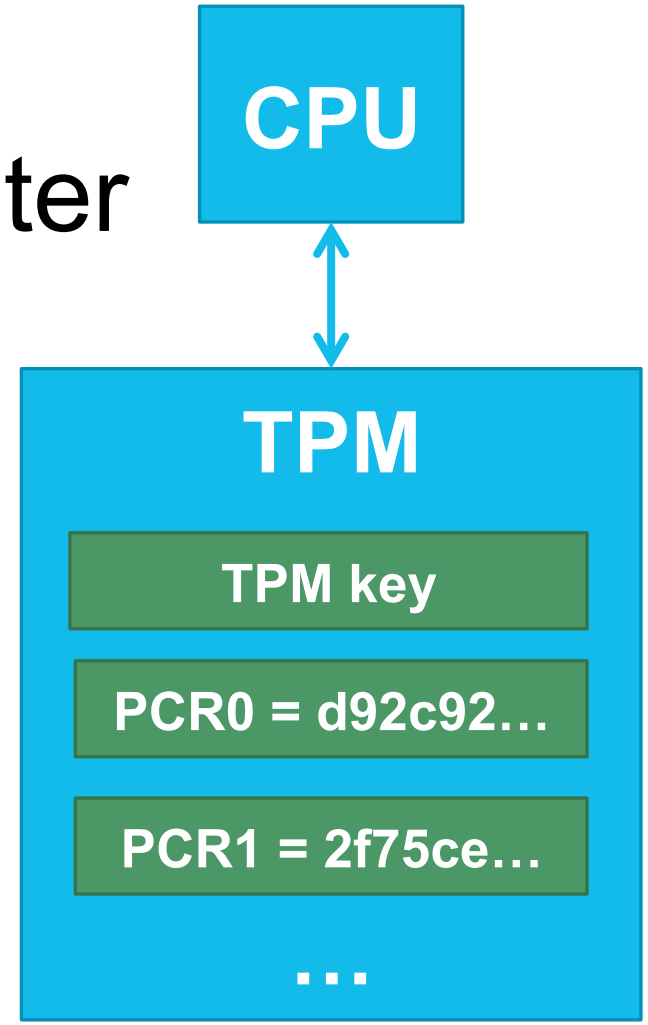
TPM interface (partial)

- PCR: platform configuration register
- TPM_extend(*n*, *m*):

  $PCR_n \leftarrow H(PCR_n \,||\, m)$
- TPM_seal(*n*, *v*, *msg*):

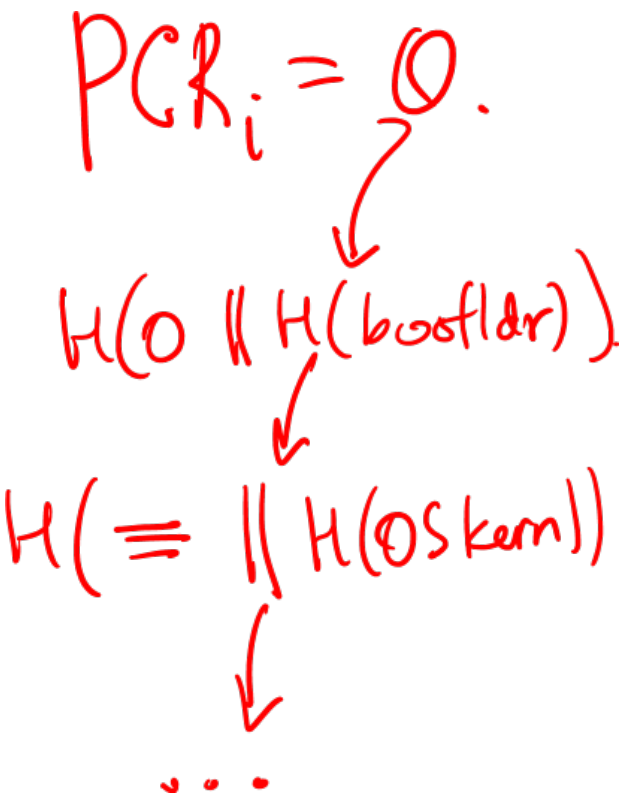
  return ciphertext
- TPM_unseal(*ciphertext*):

  return msg if $PCR_n = v$



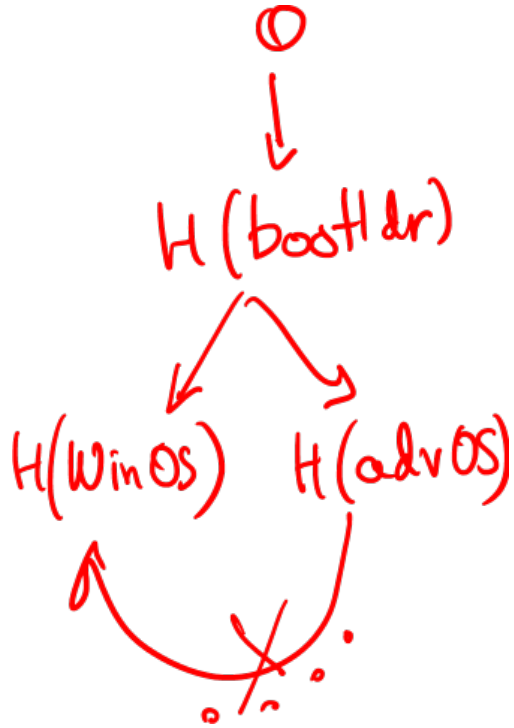
Measured boot

- PCR values reset to zero on boot
- CPU starts executing the BIOS
- Each boot step calls TPM_extend
  - BIOS: TPM_extend(H(bootloader))
  - Bootloader: TPM_extend(H(OS kernel))
  - OS kernel: TPM_extend(H(drivers))
  - …



Sealing data using the TPM

- TPM_seal(*i*, *pcr*, *msg*)
  - Can be unsealed to recover *msg* only if PCR*i* = *pcr*
  - Should mean running software corresponding to *pcr*
- What if adversary boots another OS?
  - BIOS calls TPM_extend() on adversary's bootloader
  - PCR value now differs from the expected sequence
  - Secure hash function: infeasible to find another way to get the same PCR value *pcr* via TPM_extend()
  - Will not be able to call TPM_unseal()



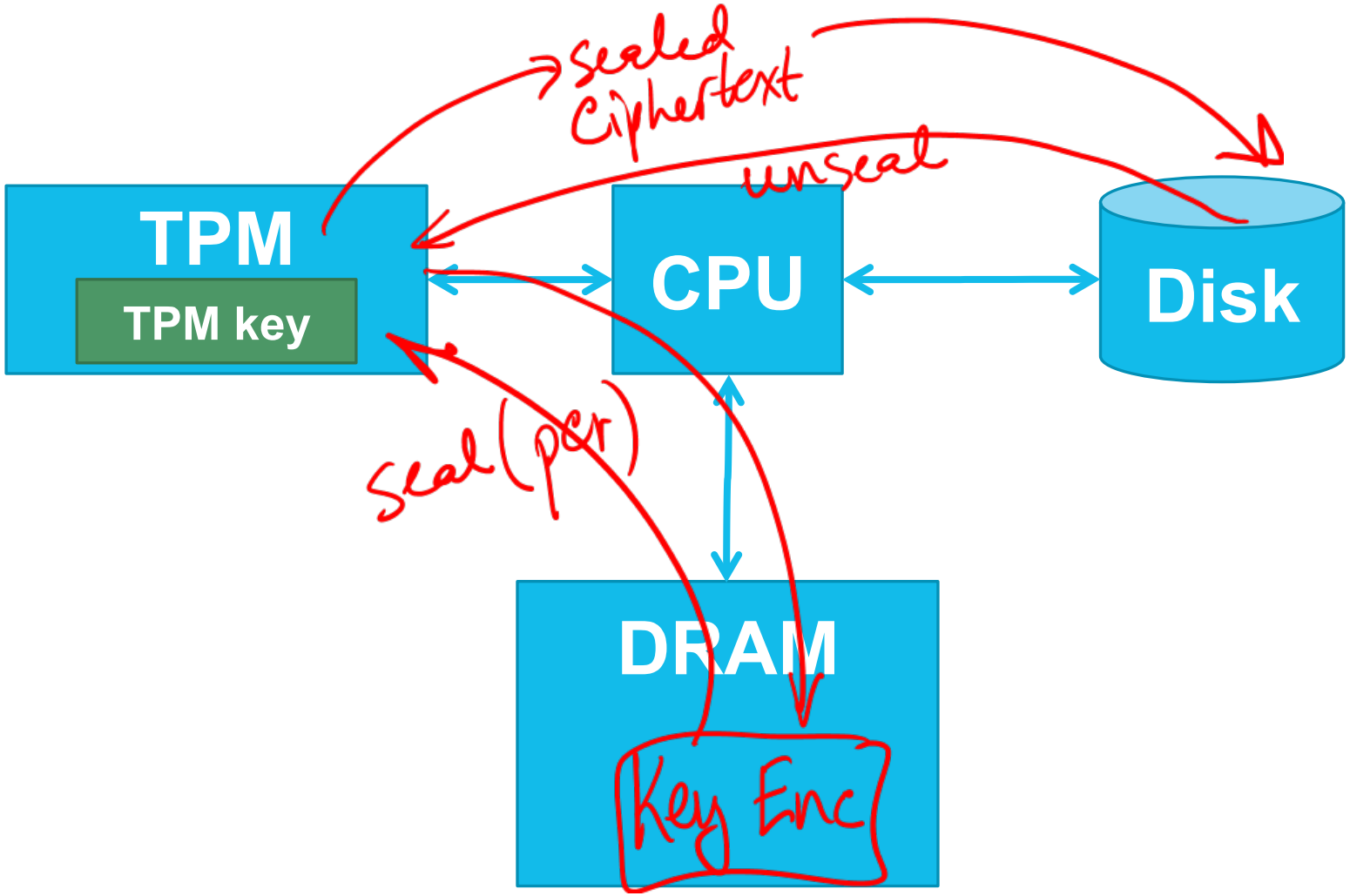
Measured boot flexibility

- Measured boot process updates PCRs to track what software was loaded
- TPM allows encrypting data that will be decrypted only under a specific PCR value
- Many benefits (and issues) of signed boot, but without hard-coding the signing key

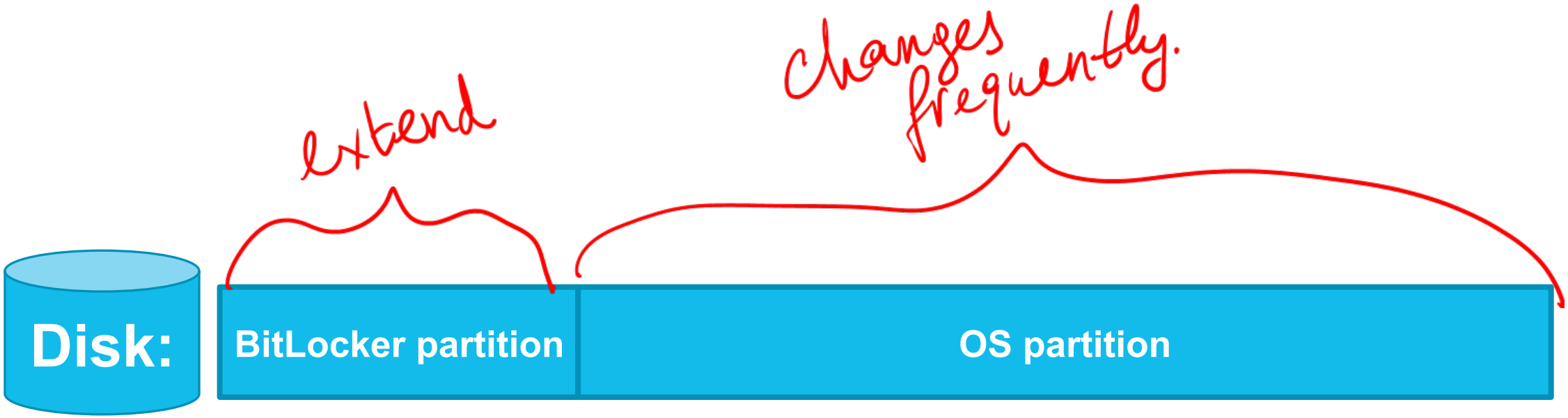
Segment 5: BitLocker keys using the TPM

- Deciding what to seal
- Deciding what to measure
- Upgrades and recovery

BitLocker seals disk encryption key

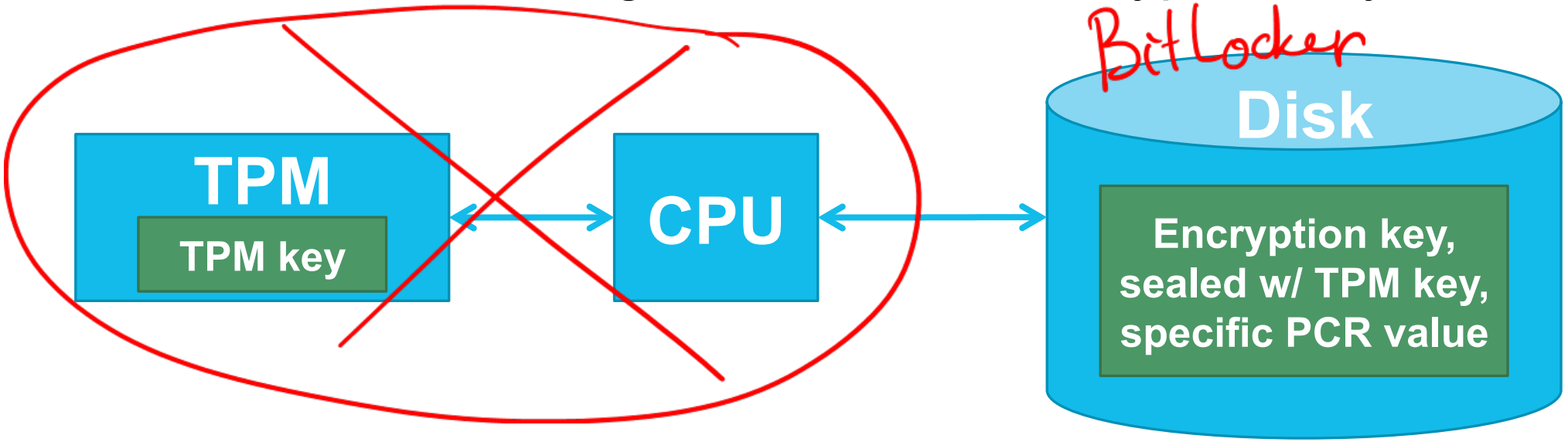


BitLocker measures small boot partition



Recovery and upgrade challenge

- Key sealed on specific TPM, for specific *pcr*
  - Upgrade BitLocker partition: PCR value changes
  - Upgrade or replace computer: TPM key changes
  - In both cases, can no longer unseal disk encryption key!



Recovery and upgrades in BitLocker

- Recovery: backup copies of disk enc. key
  - Offline USB device
  - Central server (e.g., Windows active directory)
- Upgrades: temporarily store unsealed key

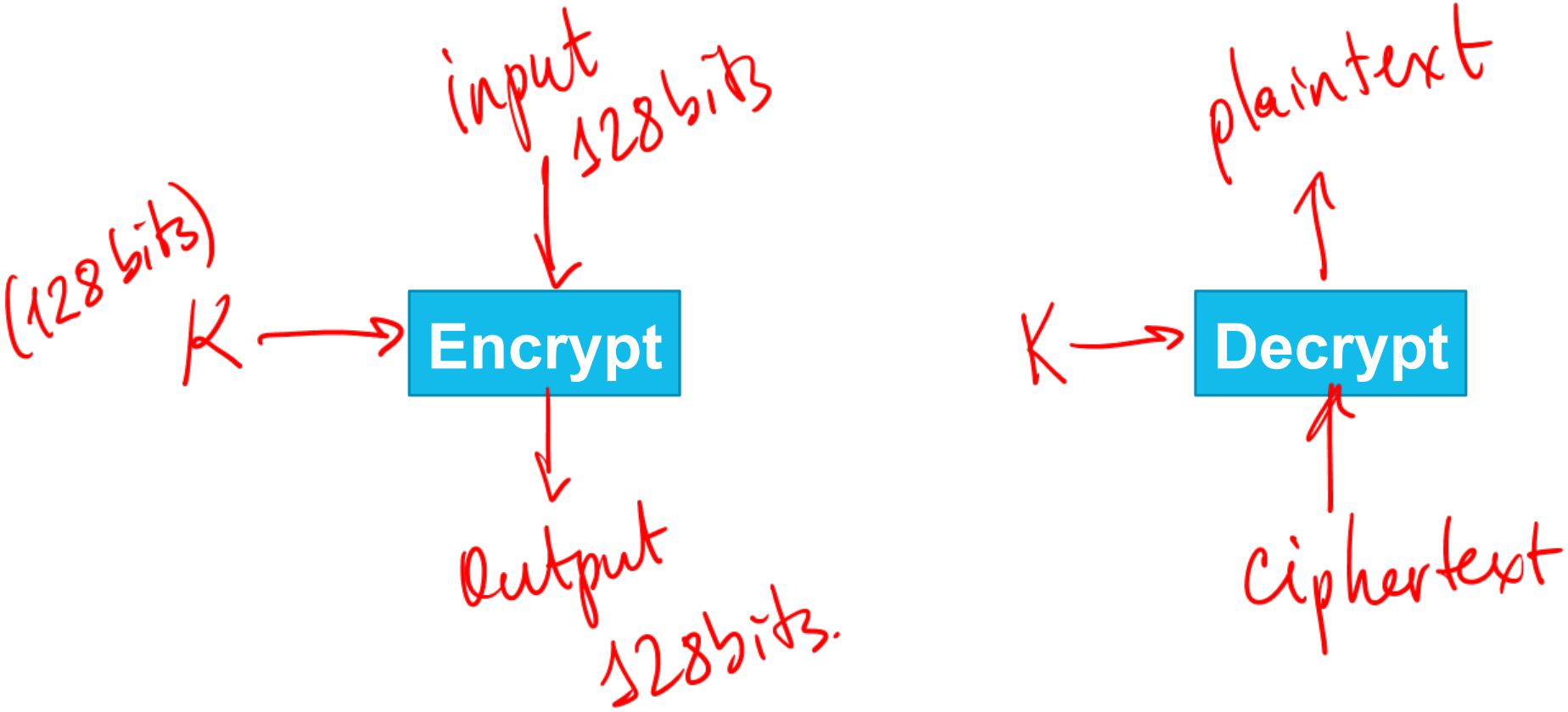
Segment 6: Data encryption

- Challenges in encrypting data
- Block ciphers
- Authentication
- Constraints of disk encryption

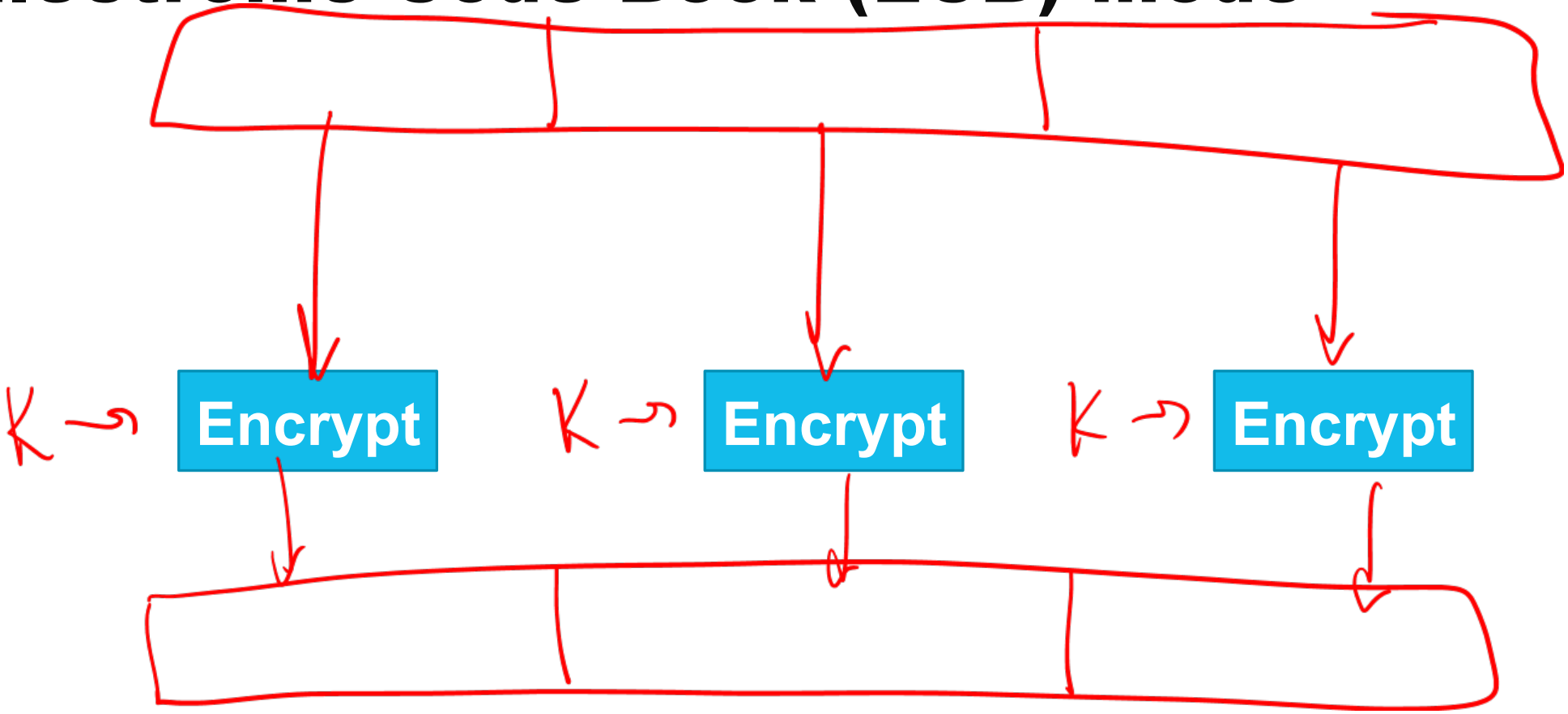
Problems with naïve encryption

1. Deterministic
2. Malleability.

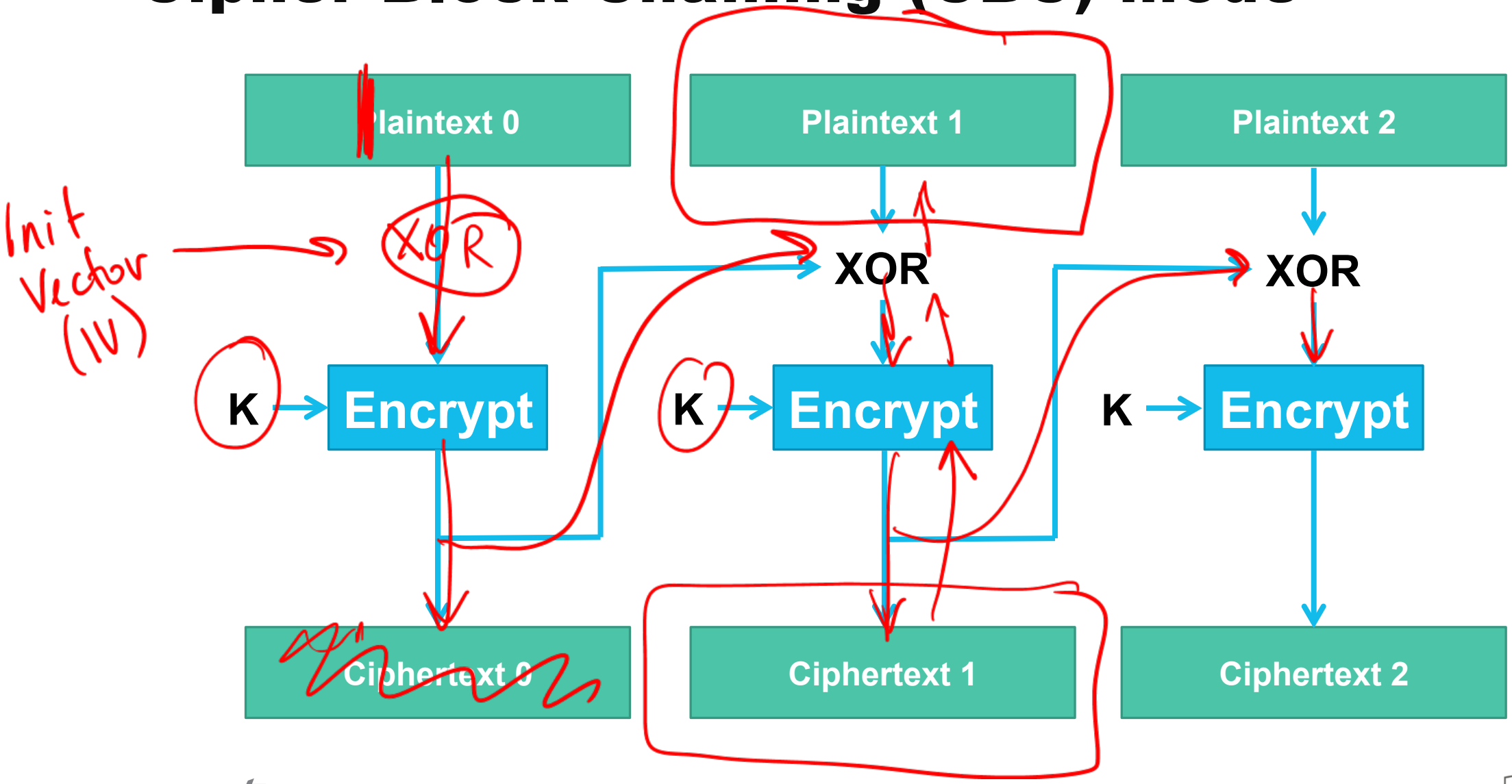
Block cipher (e.g., AES)



Encrypting more than a single AES block: Electronic Code Book (ECB) mode



Cipher Block Chaining (CBC) mode



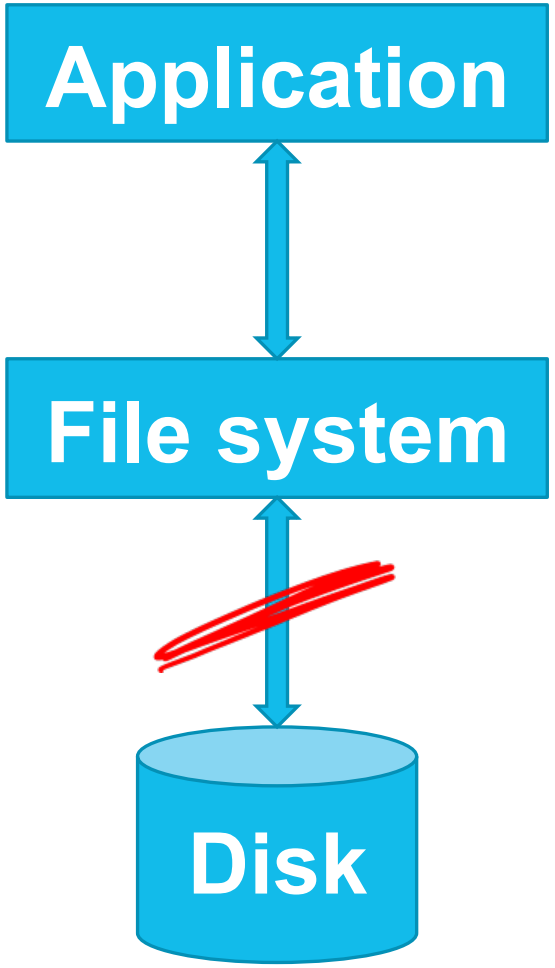
Authenticating to prevent tampering

- CBC mode: every ciphertext is “valid”
  - Cannot tell if adversary tampered with the encrypted data

- Message authentication code (MAC)
  - Rough intuition: also store MAC = H(ciphertext || key)
  - If adversary tampers with ciphertext, can’t compute H: no key
  - More efficient (and secure) constructions for authenticated encryption: UFE, OCB, CCM, …

Constraints of disk encryption

- File system expects 4KB sectors
- Disk provides 4KB sectors
- No extra space for IV or MAC!



Segment 7: BitLocker data encryption

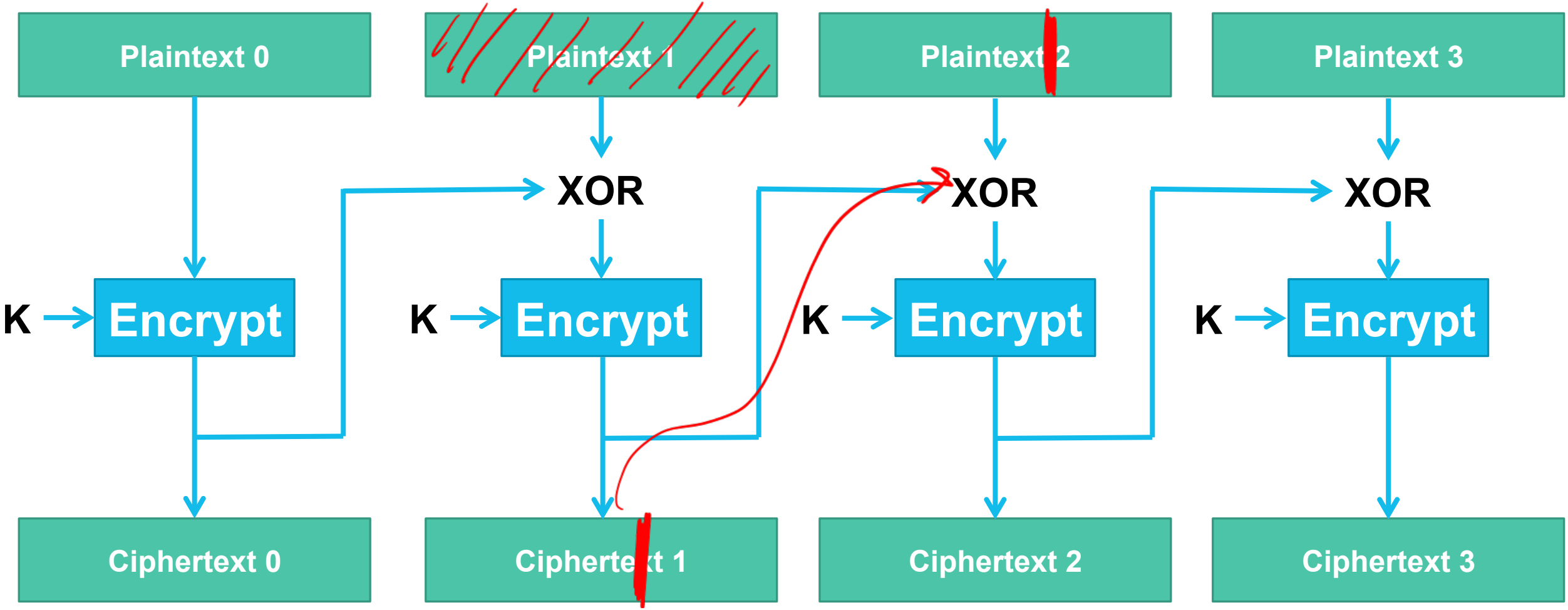
- Poor-man's authentication
- Diffusion properties of CBC
- Pseudo-random IVs
- Summary

Poor-man's authentication

- No space on disk to store per-sector MAC
- Goal: tampering with ciphertext produces "garbage" plaintext, not useful to adversary

- Good enough for our threat model
  - Adversary cannot replace login binary or password file on disk with something meaningful

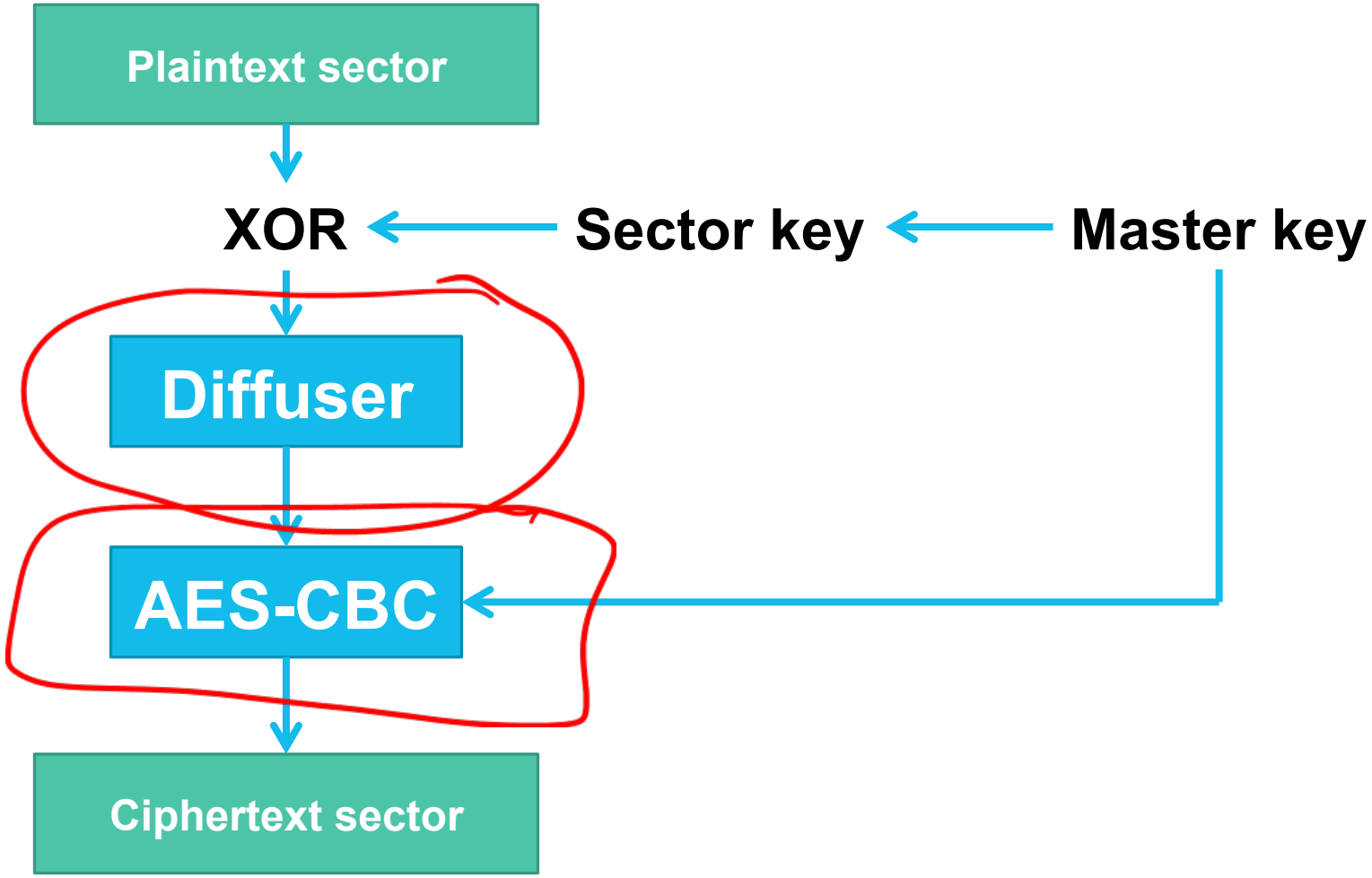
Diffusion properties of CBC



Improving diffusion with better modes

- More sophisticated block cipher modes
  - CMC, EME, XEX, …
  - Requires two passes over the data
  - Likely the right choice now
  - Was too expensive when BitLocker was designed (2006)

BitLocker's improved diffusion



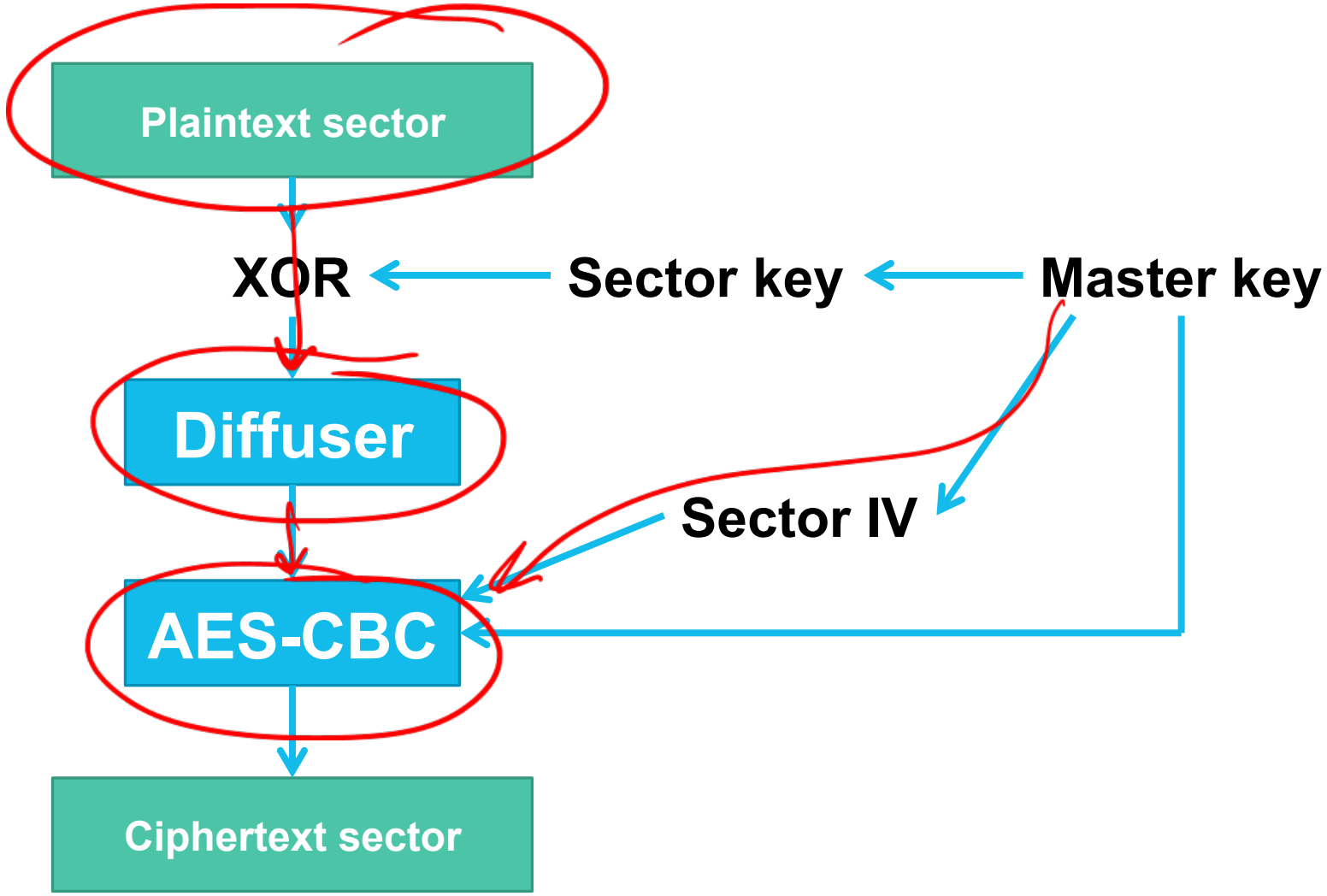
Pseudo-random IVs

- Need IV for CBC mode to randomize
- No space to store per-sector random IV
- Approach: compute pseudo-random IV

IV(sector#) = AES-Encrypt(K, sector#)

- Adversary doesn't know K, can't compute IV
  - One downside: repeated writes to same sector re-use IV

BitLocker's overall encryption plan



Summary

- Explored challenges in disk encryption
  - Key management, data encryption approaches, disk constraints
- Case study: Microsoft's BitLocker
  - Trade-offs to balance usability, performance, and security
  - Clever use of available TPM hardware
  - Poor-man's authentication deals with disk constraints
  - Careful deployment of new cryptographic constructions
  - Enables using existing Windows user authentication

THANK YOU

Nickolai Zeldovich

Associate Professor