

CS 5472 - Advanced Topics in Computer Security

Topic 5: Deniable Encryption (2)

Spring 2021 Semester

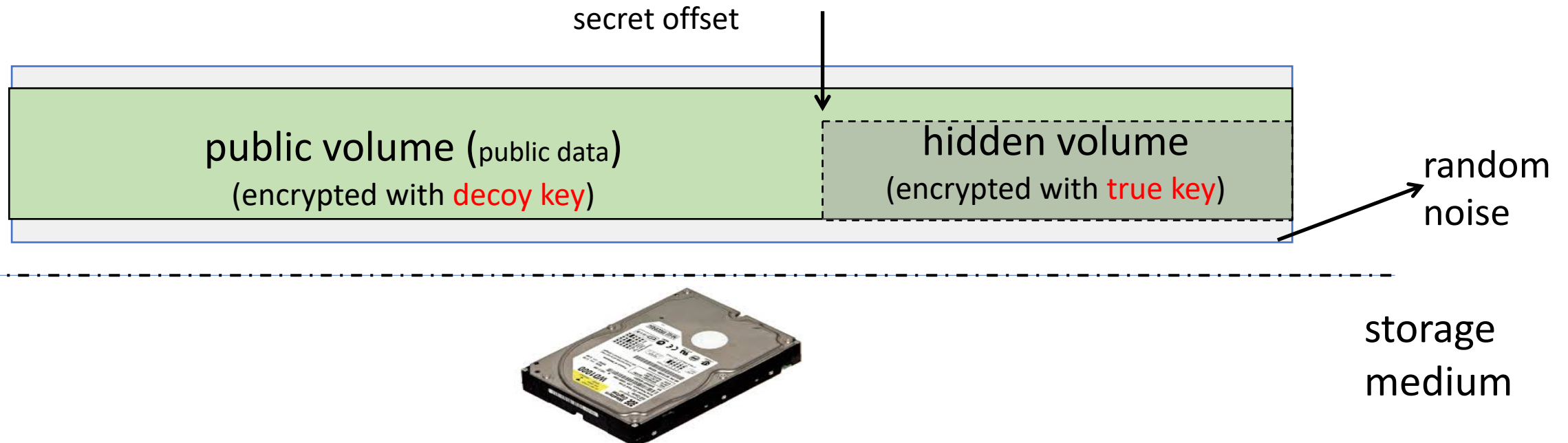
Instructor: Bo Chen

bchen@mtu.edu

<https://cs.mtu.edu/~bchen>

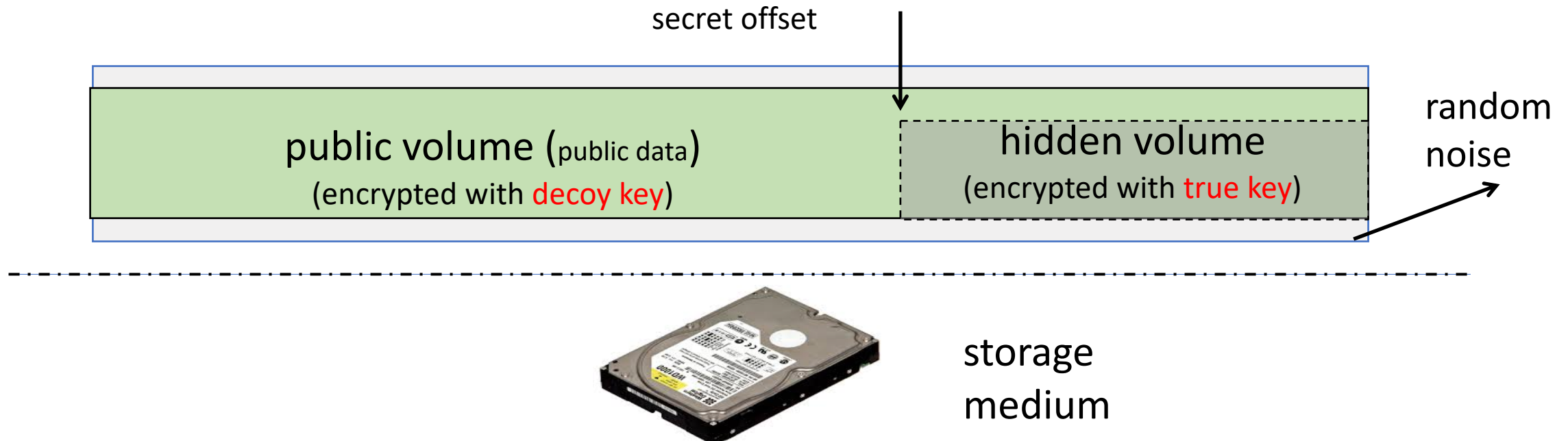
Review: Use Hidden Volume to Mitigate Coercive Attacks

- A coercive attacker can enforce the victim to disclose the decryption key
- A hidden volume-based PDE (plausibly deniable encryption) system can be used in mobile devices to mitigate coercive attacks (Mobiflage as presented on Tuesday)



Deniability Compromise 1: The Attacker Can Have Access to The Disk Multiple Times

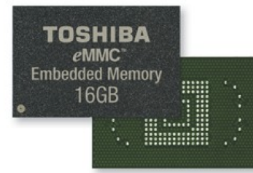
- By having multiple snapshots on the storage medium, the attacker can compromise deniability
 - Compare different snapshots and can observe the **changes/modifications over the hidden volume**, which was not supposed to happen
 - **Hidden volume is hidden in the empty space of the public volume**



Deniability Compromise 2: from The Underlying Storage Hardware

- Mobile devices usually use flash memory as the underlying storage media, rather than mechanical hard disks

- eMMC cards
- miniSD cards
- MicroSD cards



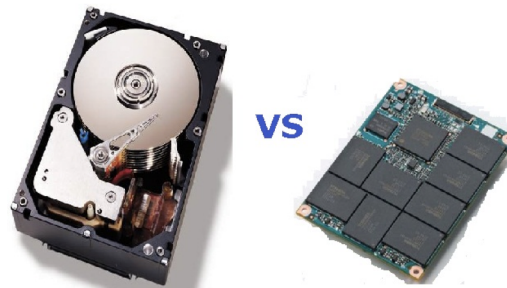
eMMC Chip



MMC Card

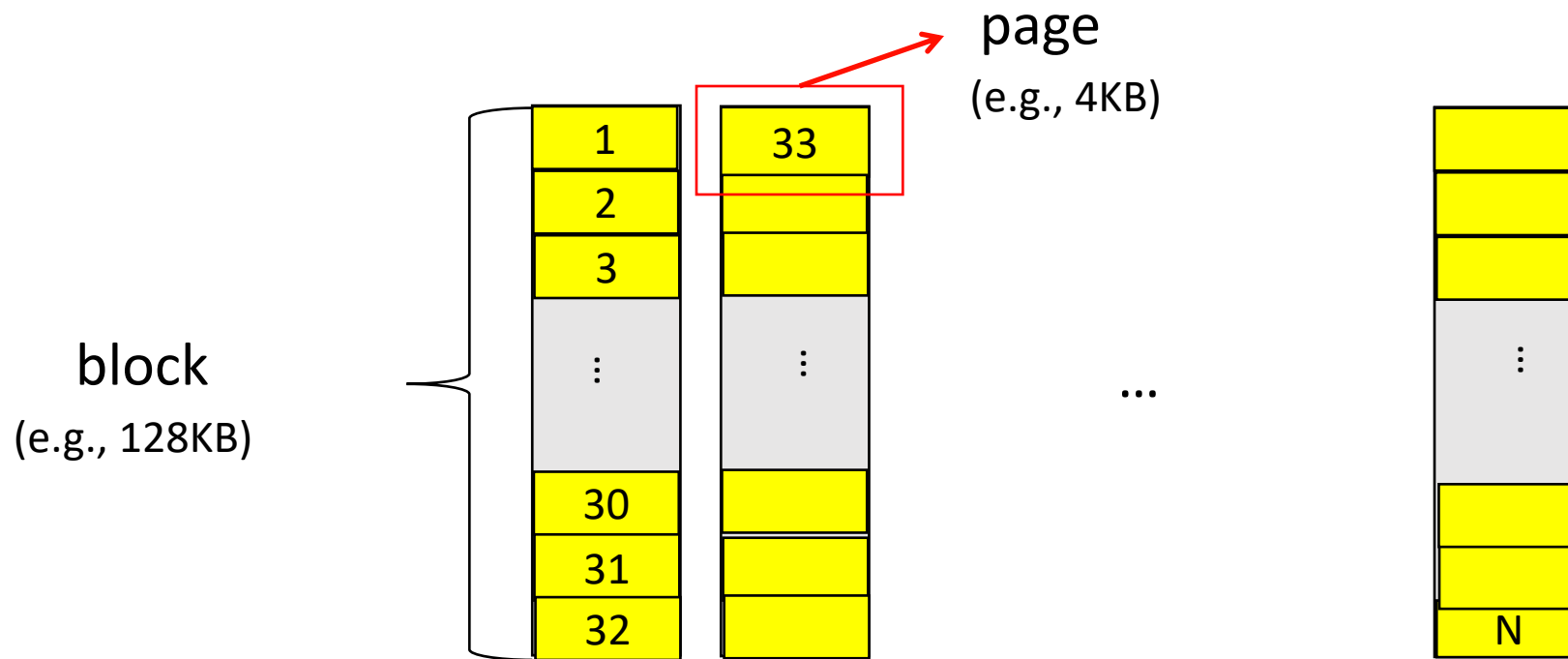
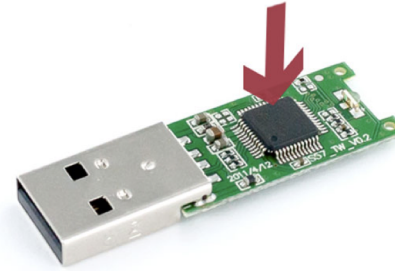


- **Flash memory has significantly different hardware nature compared to mechanical disk drives, which may cause deniability compromises unknown by the upper layers (application, file system, and block layer)**



NAND Flash is Usually Used as Storage Media

- NAND flash
 - USB sticks
 - Solid state drives (SSD)
 - SD/miniSD/microSD/eMMC



Flash Memory Programming

Write 0x0b:

Rule:

- 1) 1 can be programmed to 0
- 2) 0 cannot be programmed to 1 except performing an erasure

Write 0x0f?

Need to erase to all '1' first

All '1' initially:

1 1 1 1 1 1 1 1

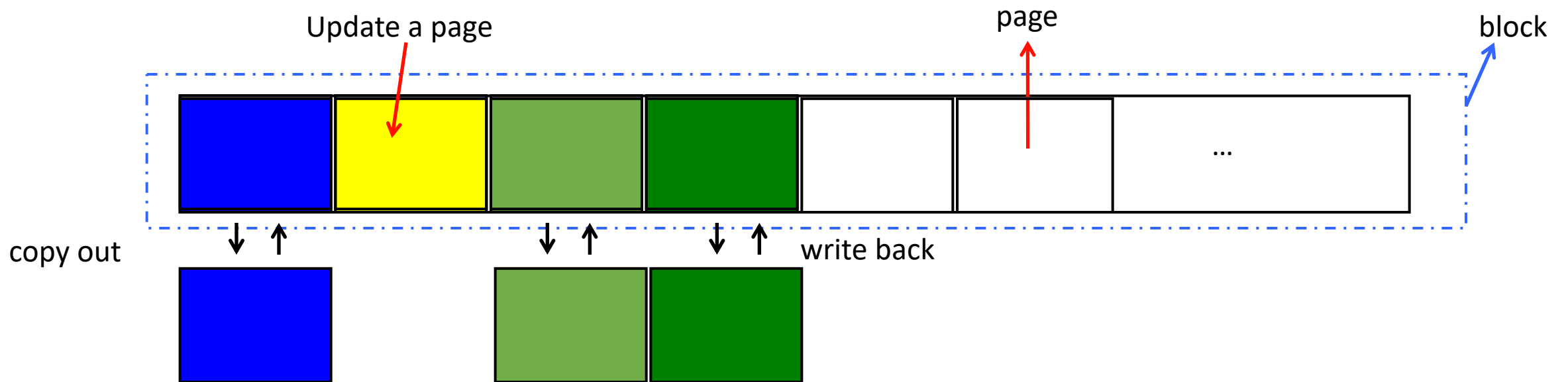
0 0 0 0 1 0 1 1

0 0 0 0 1 1 1 1

Special Characteristics of NAND Flash

- **Update unfriendly**

- Over-writing a page requires first erasing the entire block
- Write is performed in pages (e.g., 4KB), but erase is performed in blocks (e.g., 128KB)



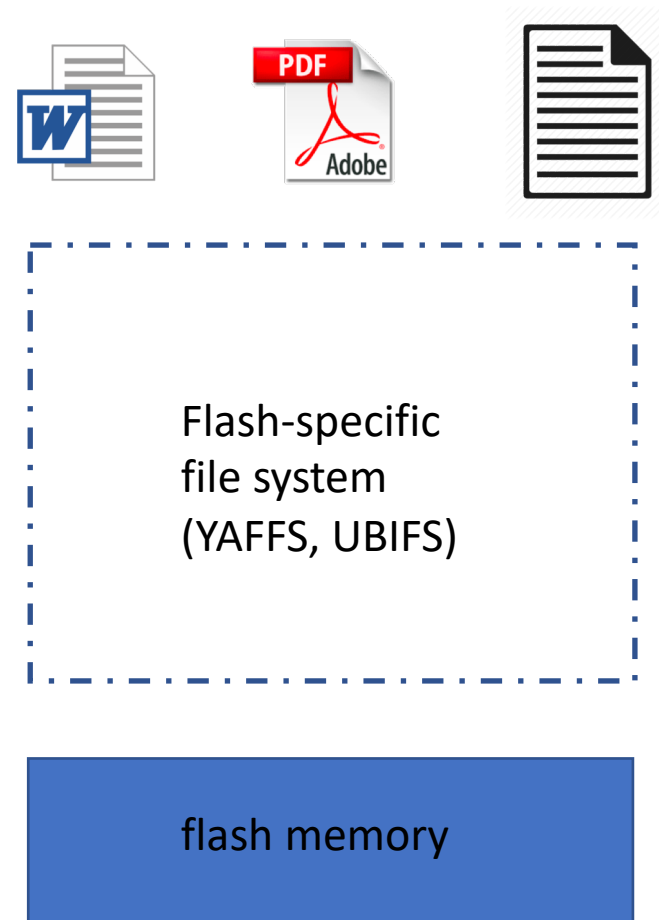
- Over-write may cause significant **write amplification**

Special Characteristics of NAND Flash (cont.)

- Support **a finite number of program-erase (P/E) cycles**
 - Each flash block can only be programmed/erased for a limited number of times (e.g., 10K)
 - Data should be placed evenly across flash (**wear leveling**)

How to Manage NAND Flash

- Flash-specific file systems, which can handle the special characteristics of NAND flash
 - YAFFS/YAFFS2, UBIFS, F2FS, JFFS/JFFS2
 - Less popular

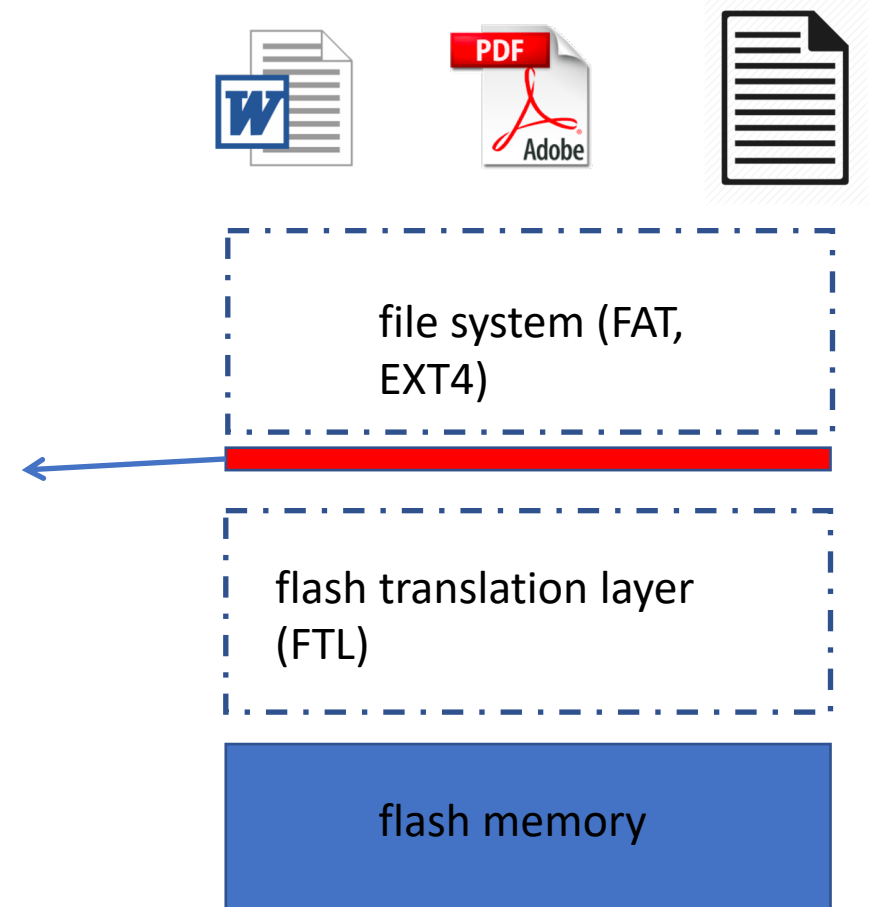


How to Manage NAND Flash (cont.)

- Flash translation layer (FTL) – a piece of flash firmware embedded into the flash storage device, which can handle the special characteristics of NAND flash and emulate the flash storage as a regular block device (**most popular**)
 - SSD
 - USB
 - SD

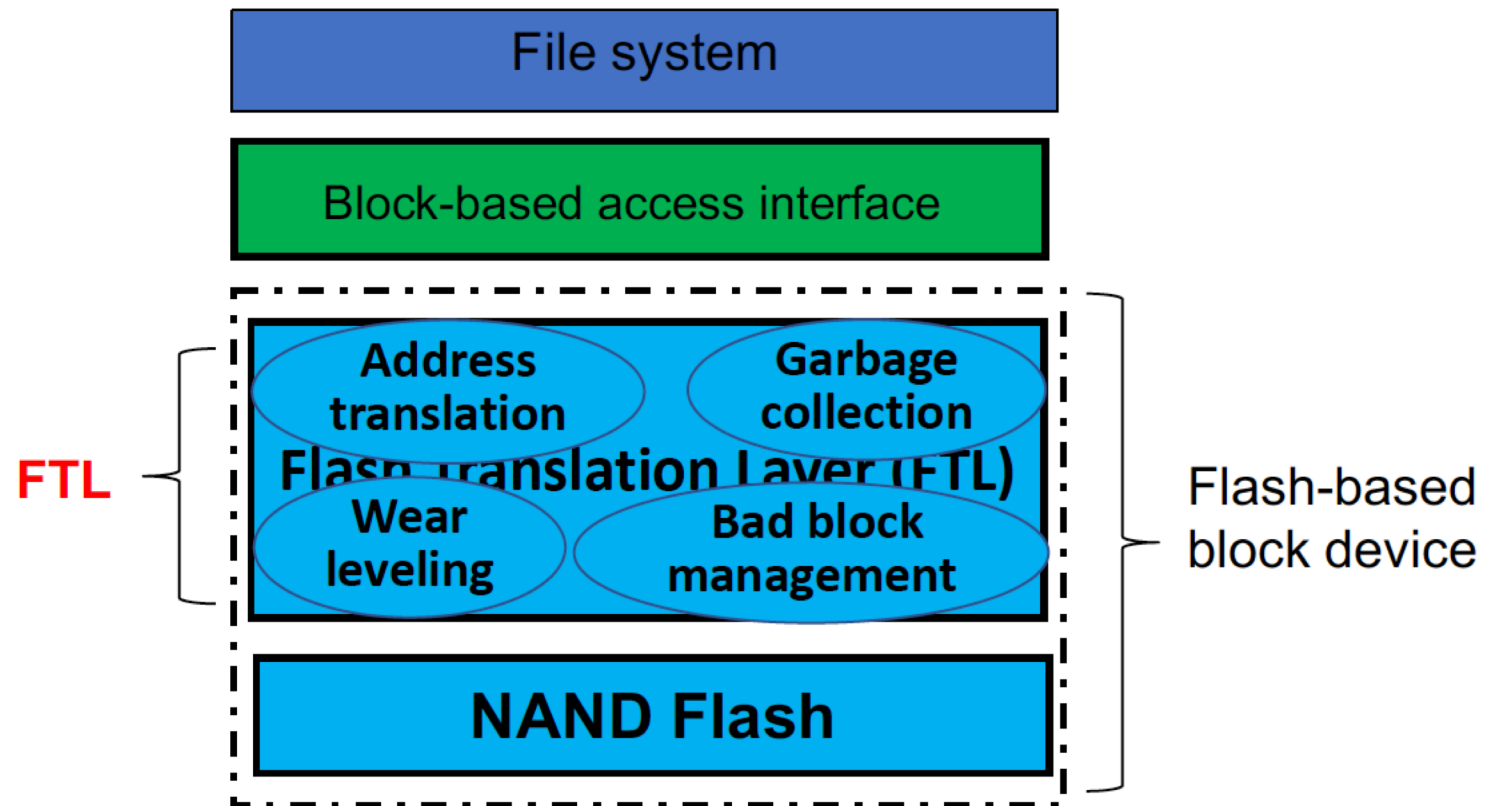


block device
interface:



Flash Translation Layer (FTL)

- FTL usually provides the following functionality:
 - Address translation
 - Garbage collection
 - Wear leveling
 - Bad block management



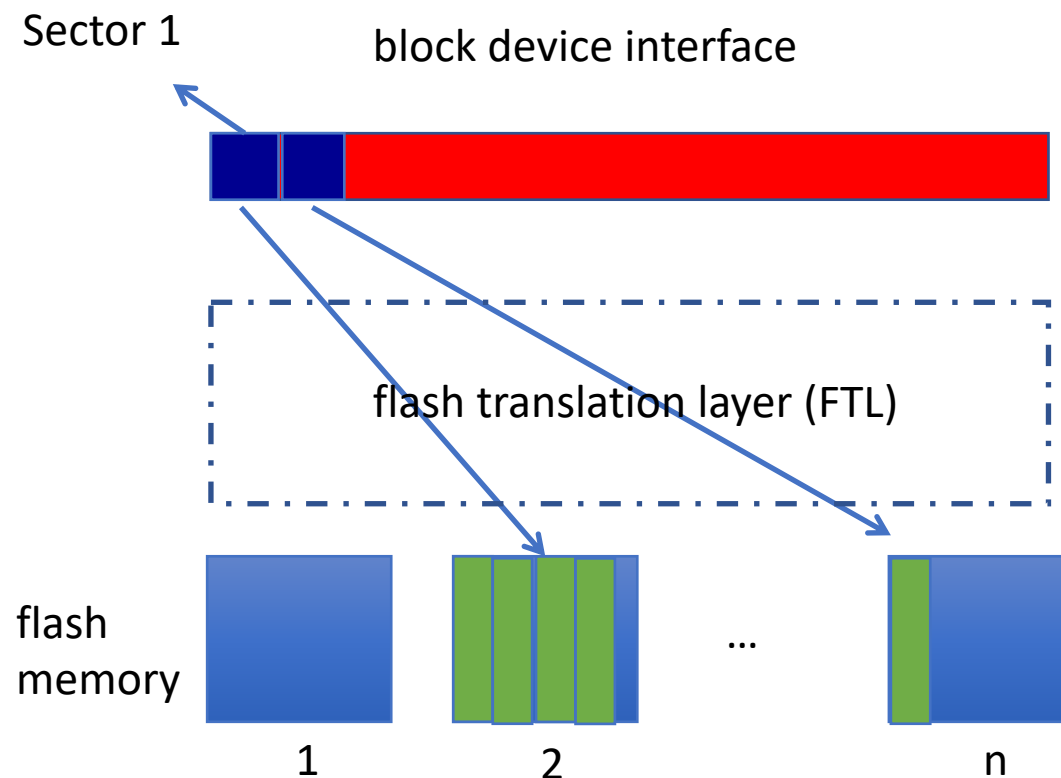
Flash Translation Layer (cont.)

FTL should maintain a mapping table

Block device location	Flash location
Sector 1	(2,3)
Sector 2	(n,1)

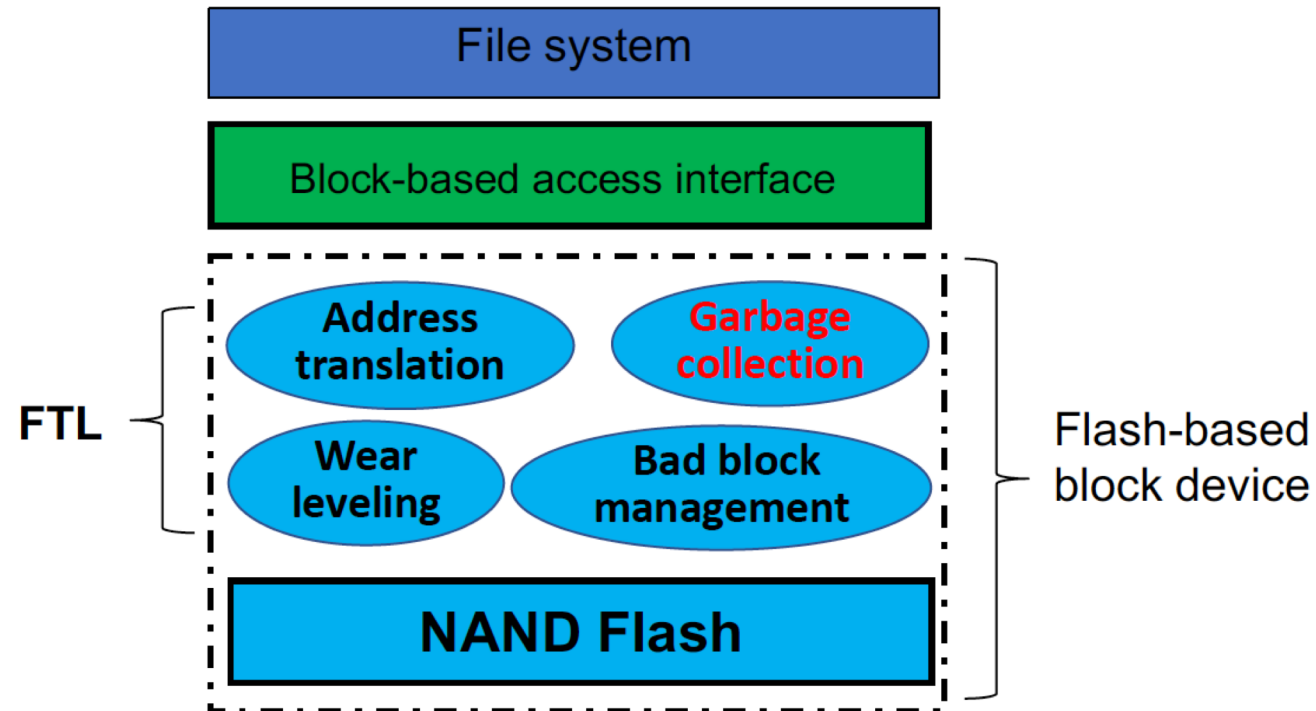
- Address translation

- Translate address between block addresses and flash memory addresses
- Need to keep track of mappings between Logical Block Address (LBA) and Physical Block Address (PBA)



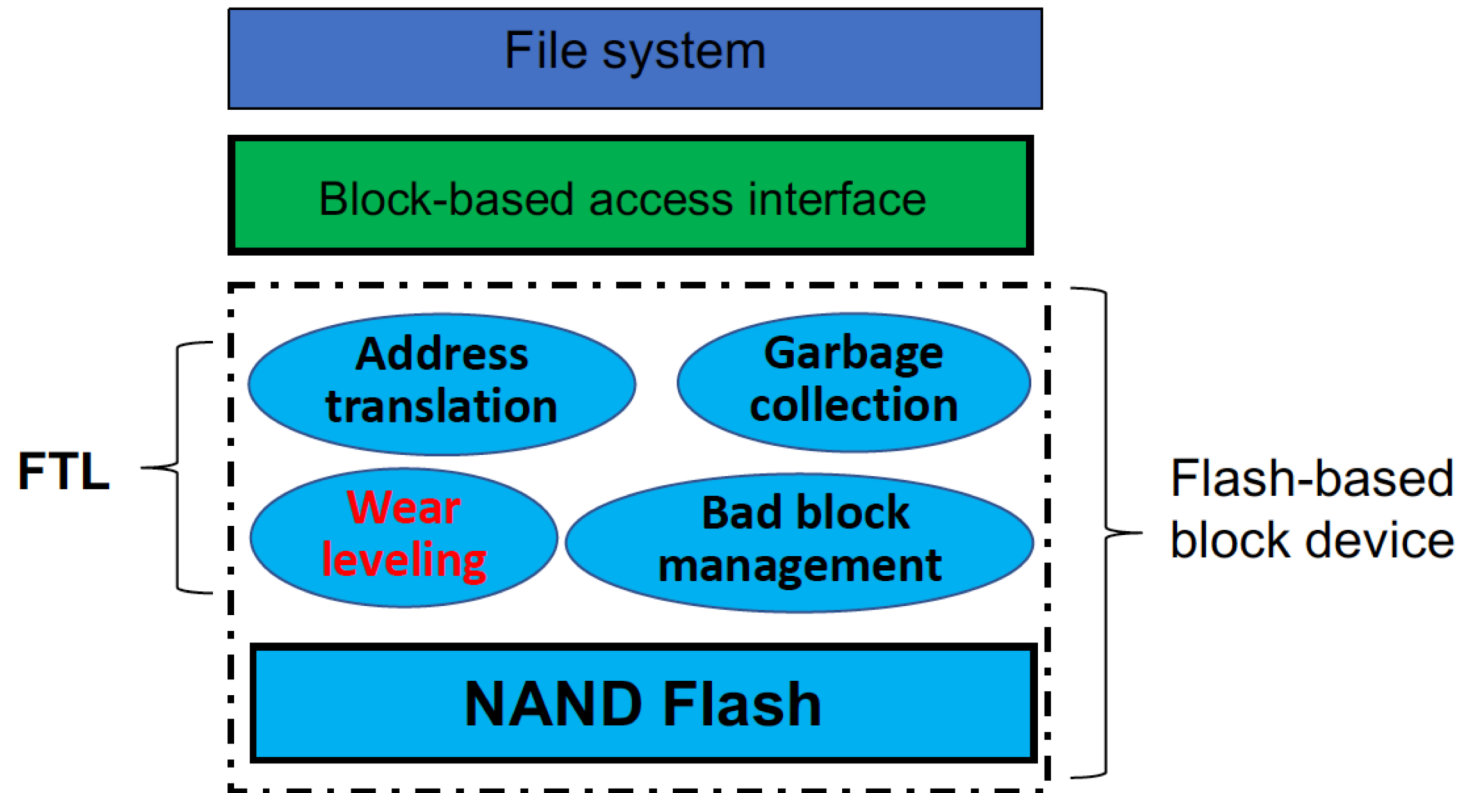
Flash Translation Layer (cont.)

- Garbage collection
 - Flash memory is update unfriendly
 - Not prefer in-place update, but prefer out-of-place update
 - The blocks storing obsolete data should be reclaimed periodically by garbage collection



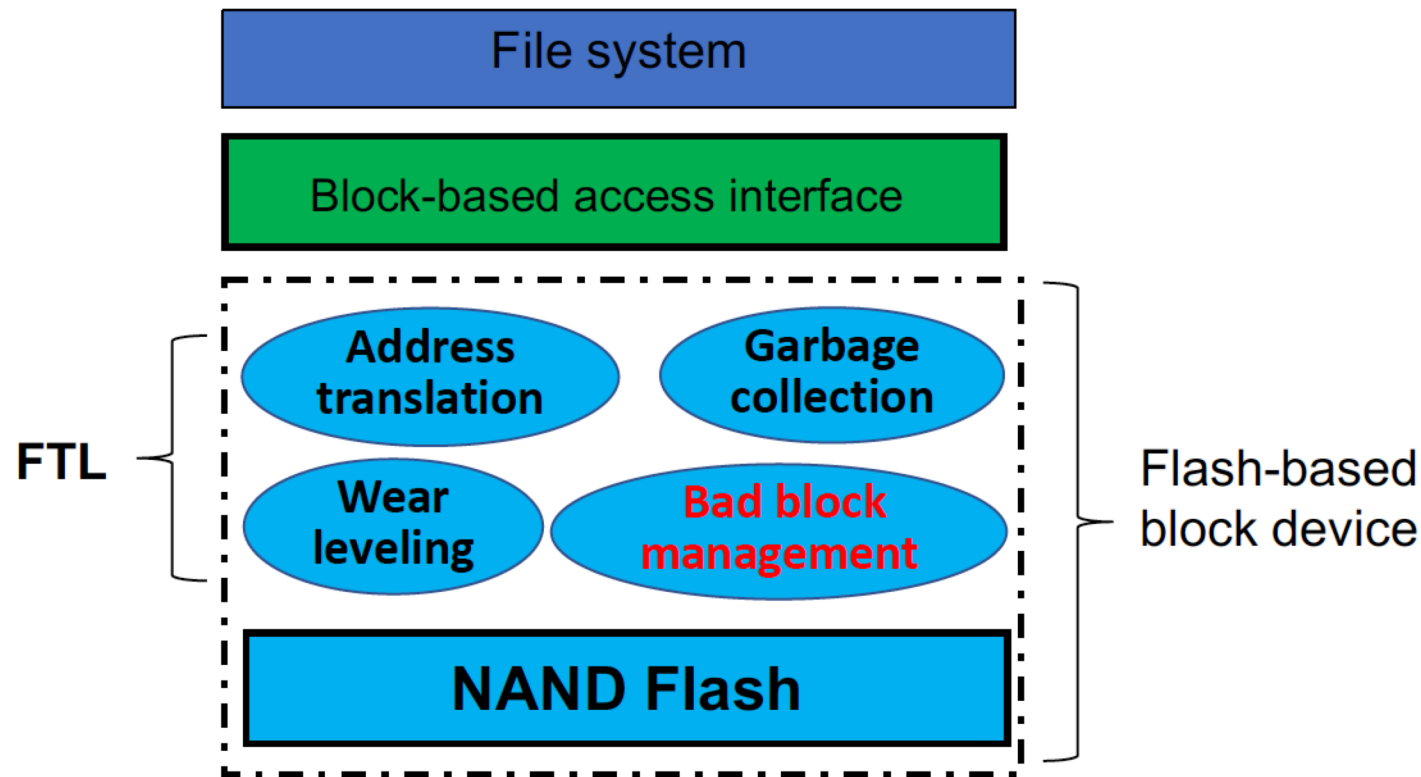
Flash Translation Layer (cont.)

- Wear leveling
 - Each flash block can be programmed/erased for a limited number of times
 - Distribute writes evenly across the flash to prolong its lifetime

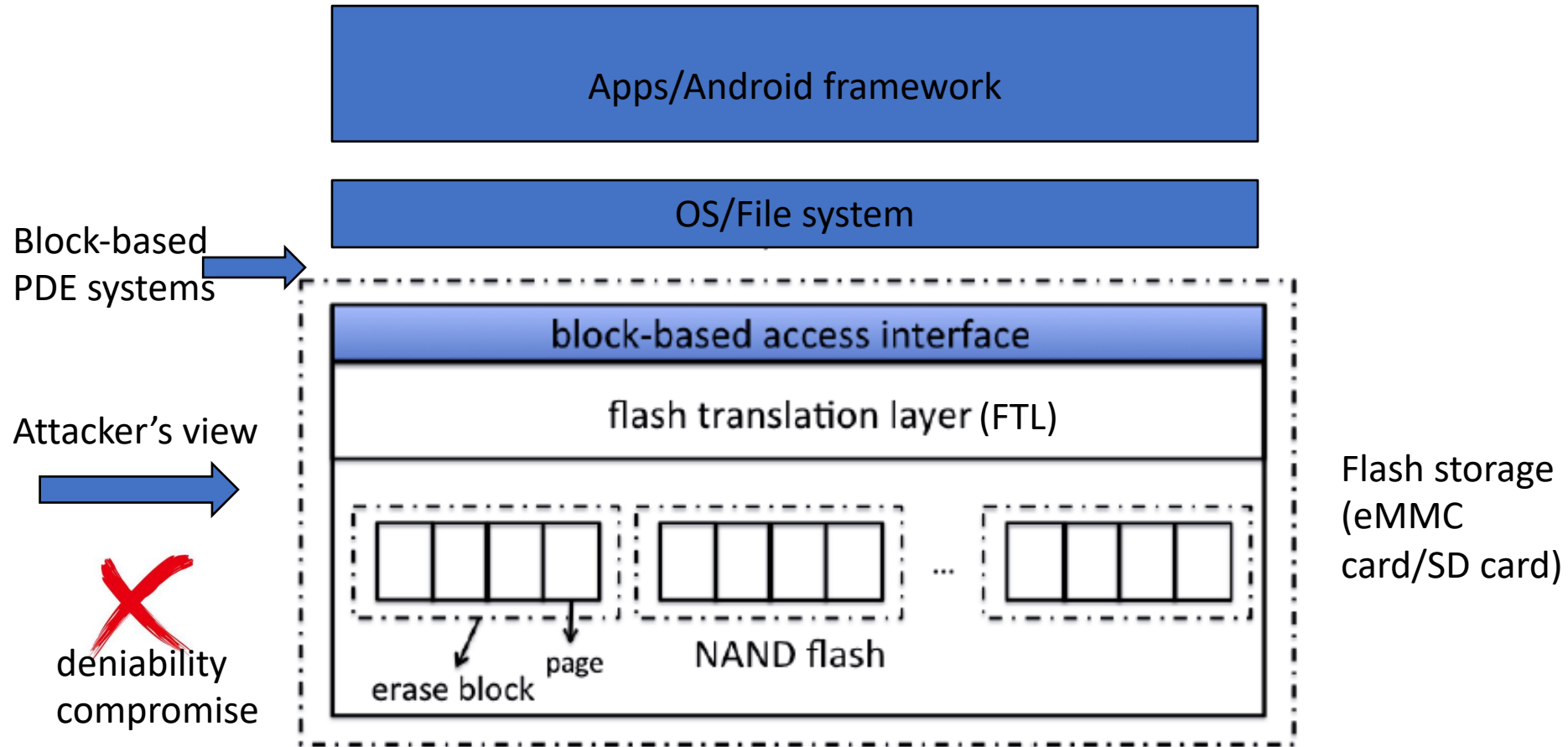


Flash Translation Layer (cont.)

- Bad block management
 - Regardless how good is the wear leveling, some flash blocks will eventually turn “bad” and cannot reliably store data
 - Bad block management is to manage these bad blocks



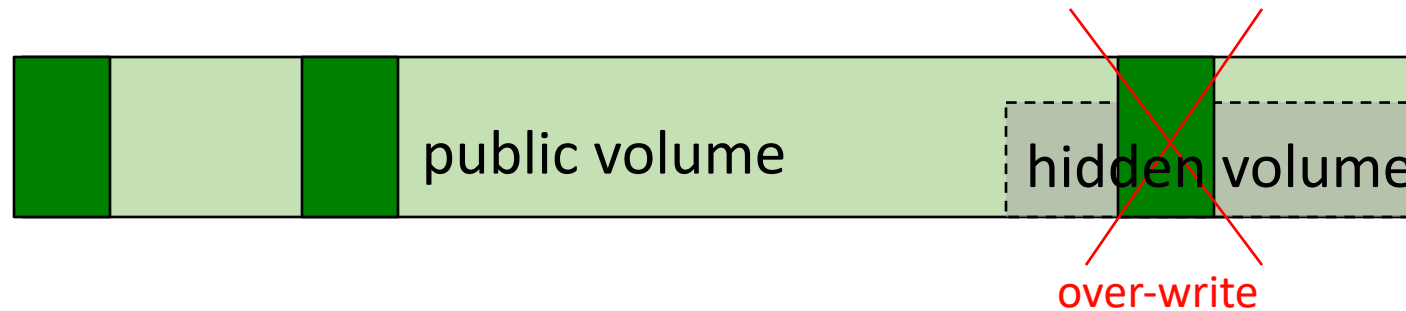
Deniability May be Compromised When Deploying Hidden Volume on The Block Layer



By obtaining a view in the flash memory, the adversary may be able to observe those unexpected “traces” of the hidden sensitive data (The trace are due to handling the special nature of ash memory)

Overwrite Issues Faced by The Hidden Volume Technique

- The data written to the public volume (if not written sequentially) may **over-write** the data in the hidden volume
 - The hidden volume is part of the public volume



Paper Presentation

- DEFTL: Implementing Plausibly Deniable Encryption in Flash Translation Layer
- Presented by Niusen Chen (guest presenter)

DEFTL: Implementing Plausibly Deniable Encryption in Flash Translation Layer

Shijie Jia, Luning Xia, Bo Chen, Peng Liu

Presenter: Niusen Chen (PhD student)
Department of Computer Science
Michigan Technological University

Outline

- Background Introduction
- Attack Scenarios
- Design of DEFTL
- Evaluation

Mobile Devices is Ubiquitous



Smartphone

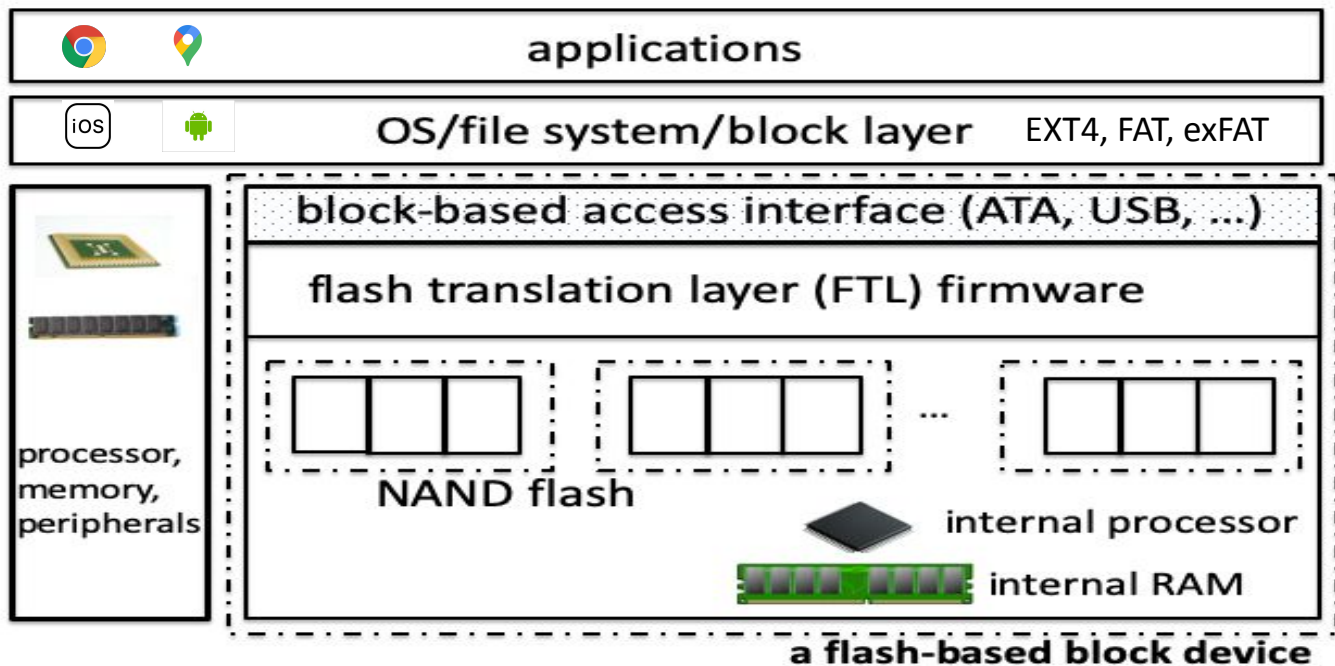


Tablet



Smartwatch

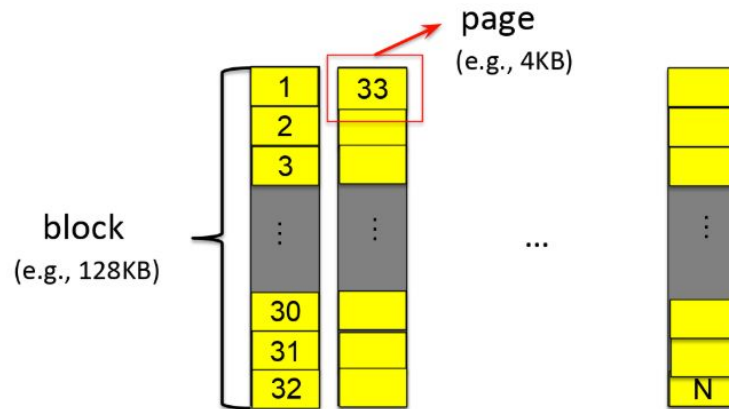
The Mainstream Architecture of Mobile Device



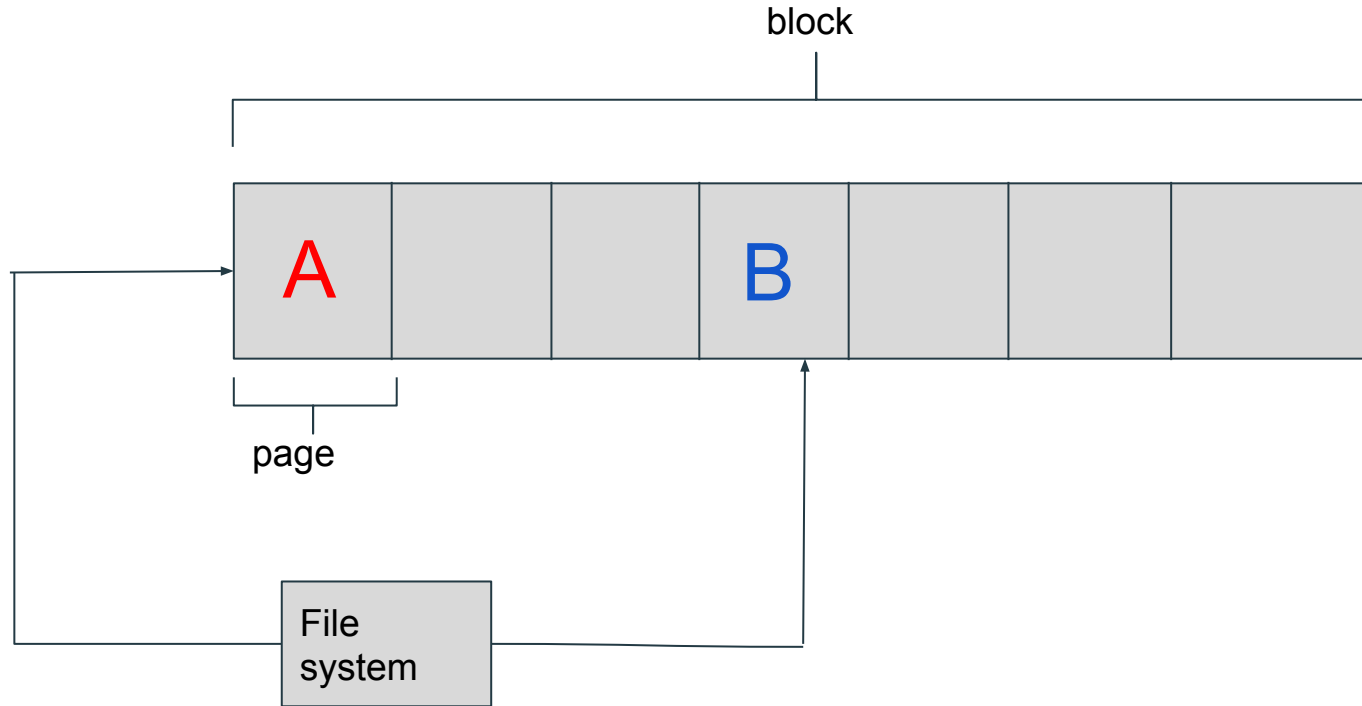
Flash-based block device: A flash-based storage device that supports reading and (optionally) writing data in fixed-size blocks, sectors, or clusters. These blocks are generally 512 bytes

Features of Flash Memory

1. Read/Write on pages, but erase on blocks
2. Erase - before - write
3. Out - of - place update
4. Limited of program-erase(P/E) cycles

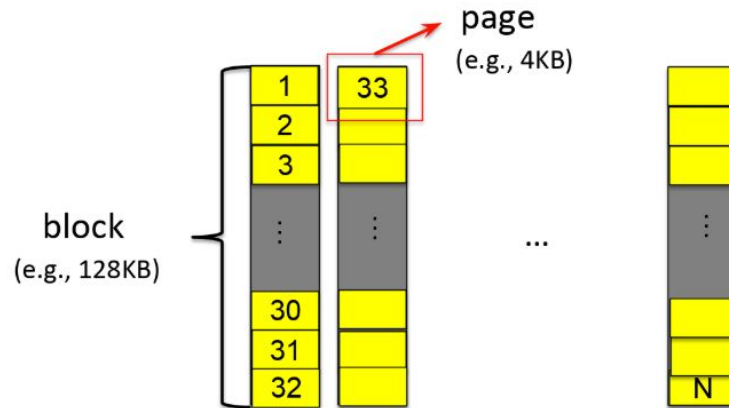


Update: Replace value “A” with “B” in flash memory



Features of Flash Memory

1. Read/Write on pages, but erase on blocks
2. Erase - before - write
3. Out - of - place update
4. Limited of program-erase(P/E) cycles



Special Functions in Flash

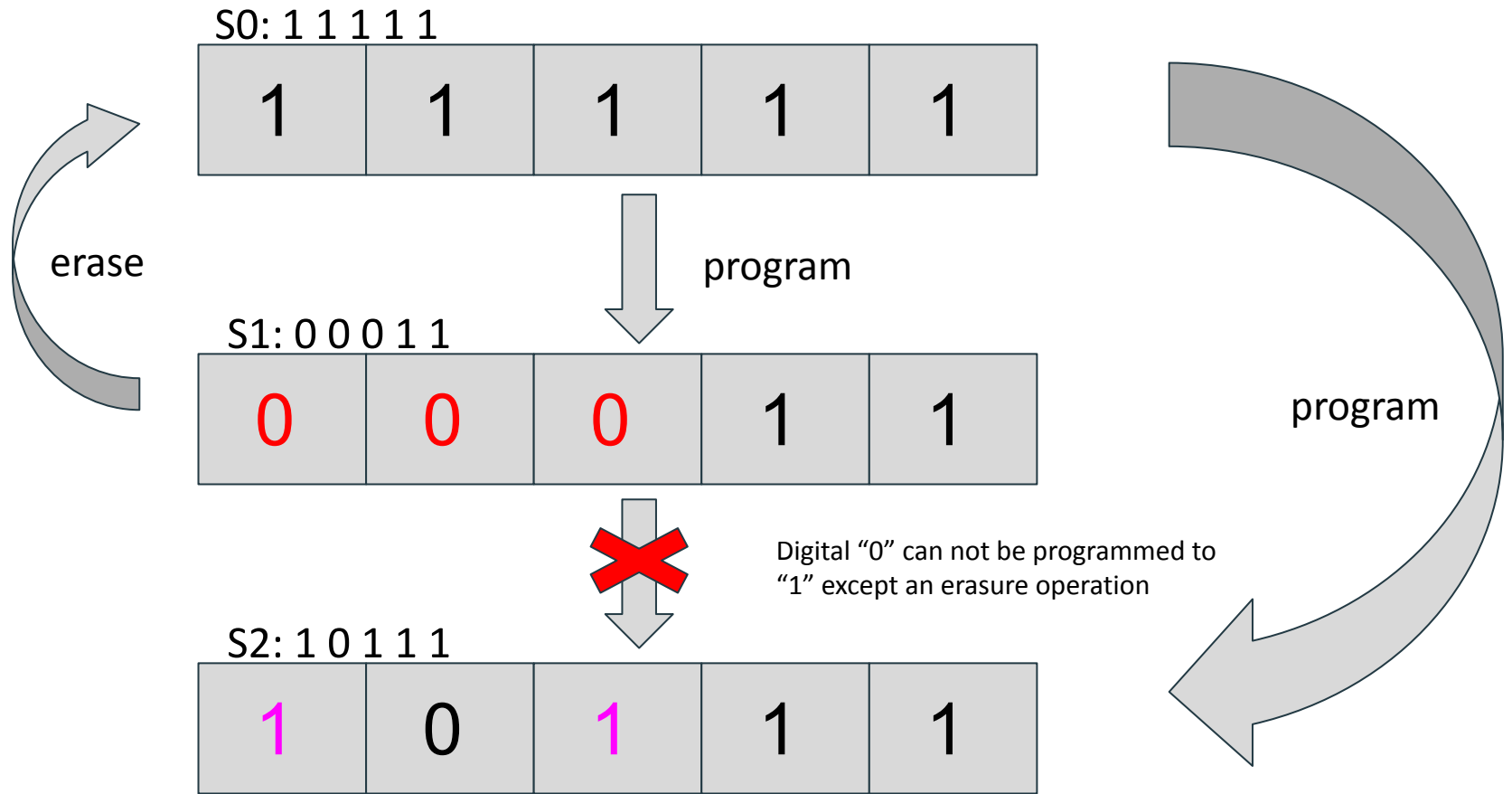
Garbage Collection: Blocks containing too many invalid pages will be reclaimed by copying valid data out of them, and the reclaimed blocks will be placed to free block pool to be re-used

Wear Levelling: Distribute writes/erasures evenly across flash memory by swapping hot and cold blocks

How to Program/Write Data to Flash

Three rules:

- Initially, what in flash memory are all digital “1”s
- Digital “1” can be programmed to digital “0” (write operation)
- Digital “0” can not be programmed to “1” except an erasure operation



How to Use Flash

File System (FAT, EXT4)

Flash Translation Layer
(FTL)

Flash Memory

Method 1: FTL

SanDisk



Goldenfir



Flash-specific File System
(YAFFS, UBIFS)

Flash Memory

Method 2: Flash File System

Full Disk Encryption (FDE)

1. Everything on disk is encrypted
2. Totally transparent to users
3. Can not defend against coercive attack

Coercive Attack:

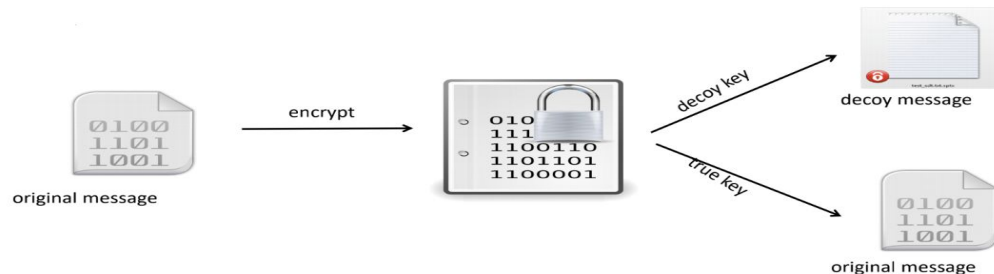
An attacker forces the device owner to disclose the decryption key



FDE is vulnerable to a coercive attack

Plausibly Deniable Encryption (PDE):

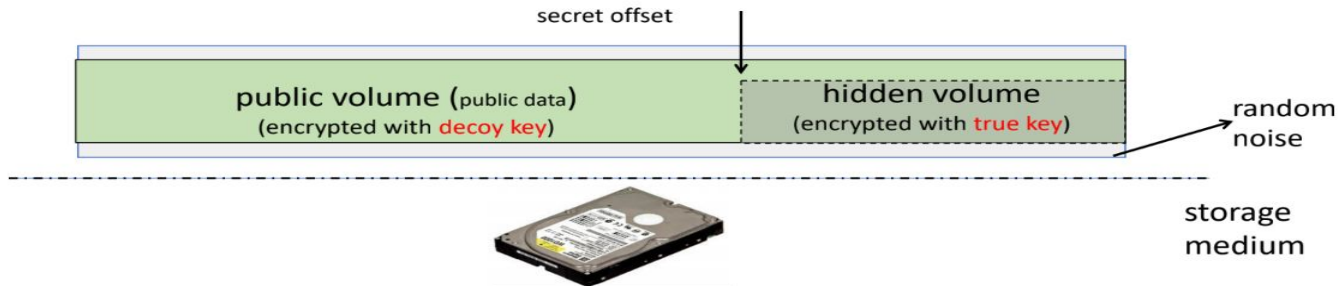
- A crypto primitive designed for mitigating coercive attacks
- Plain text is encrypted by a true key and a decoy key such that:
 - Decrypt with decoy key ➡ Decoy message
 - Decrypt with true key ➡ True message
- Upon being coerced: disclose decoy key, keep true key
- PDE is hard to be achieved in crypto
- Two techniques to simulate PDE
 - Hidden volume technique
 - Steganography technique



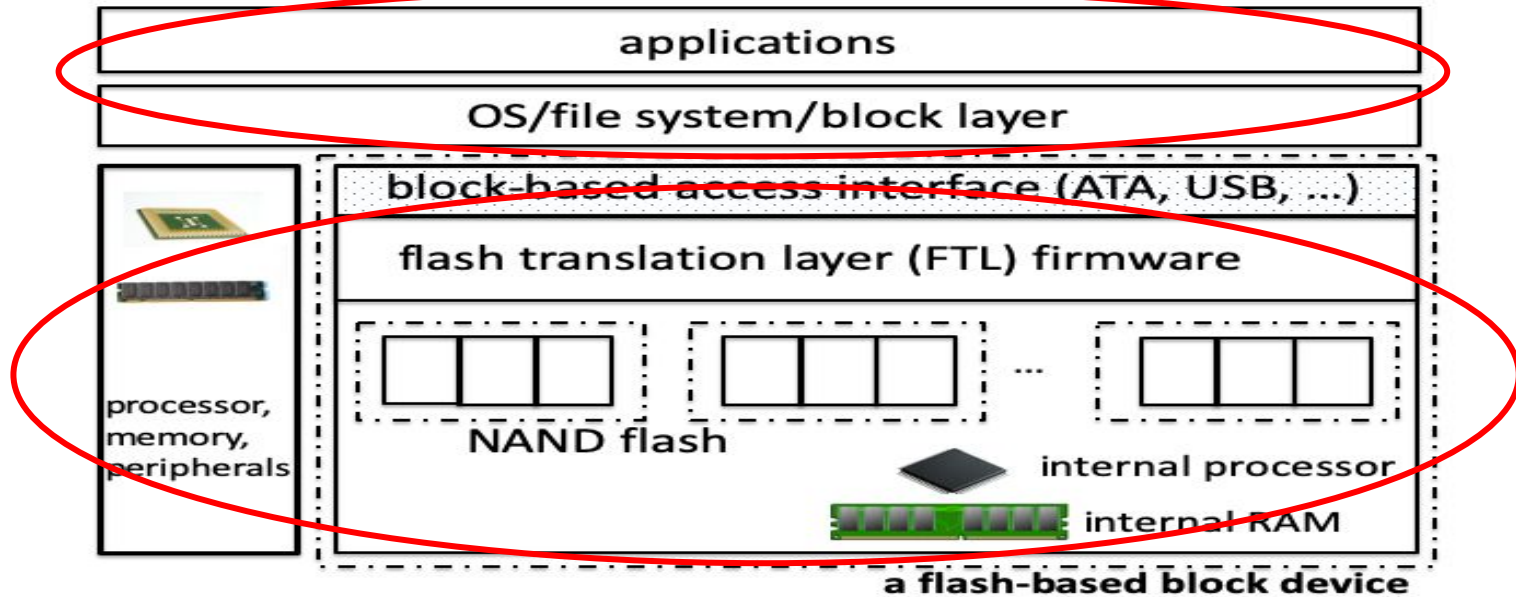
PDE Technique (cont.)

Hidden Volume Technique

- Whole disk is initialized with randomness
- Two volumes: public volume and hidden volume
 - Public volume: encrypted with a **decoy** key; store non-sensitive data
 - Hidden volume: encrypted with **true** key; store sensitive data
- Disclosing the decoy key upon being coerced by attacker

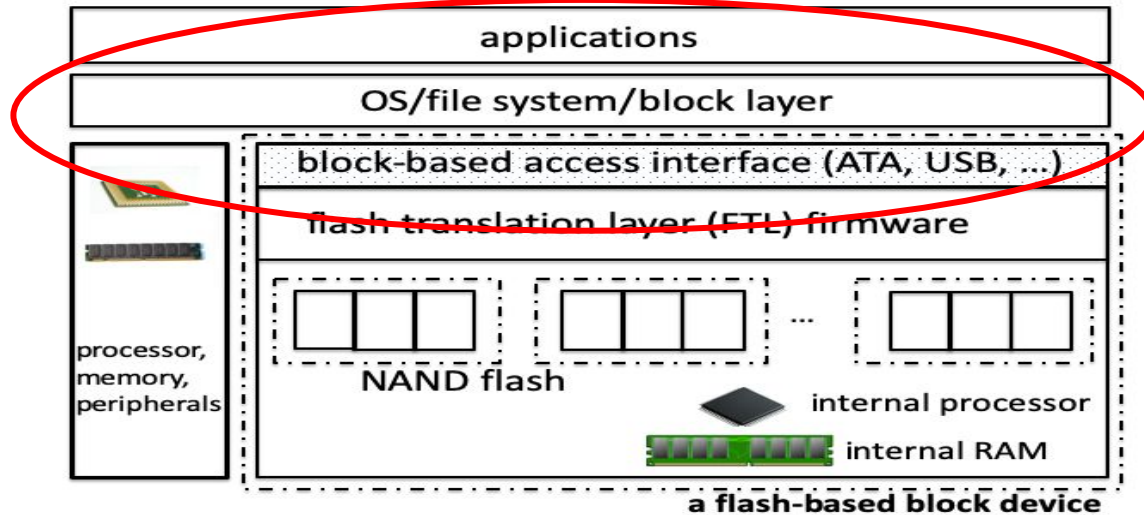


How to Build PDE in Mobile Device



- Build PDE in application/block layer
- Build PDE in FTL layer
- Design a file system supporting PDE

How to Build PDE in Mobile Device (cont.)



	Layer
MobiFlage	Block device
MobiHydra	Block device
MobiPluto	Block device
DEFY	File system

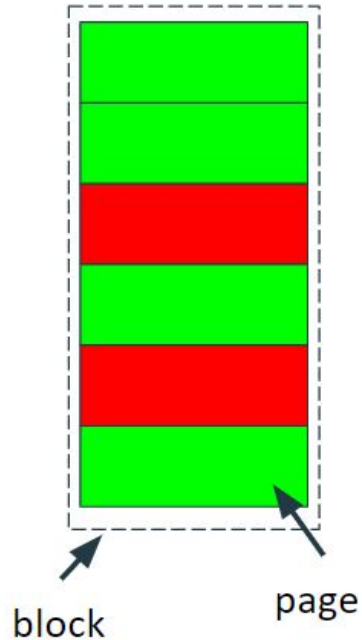
- Build PDE in application/block layer
- Design a file system supporting PDE
- Build PDE in FTL layer

Deniability could be compromised in FTL layer

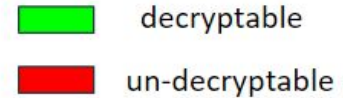
Attack Scenario

Attack 1

With hidden volume

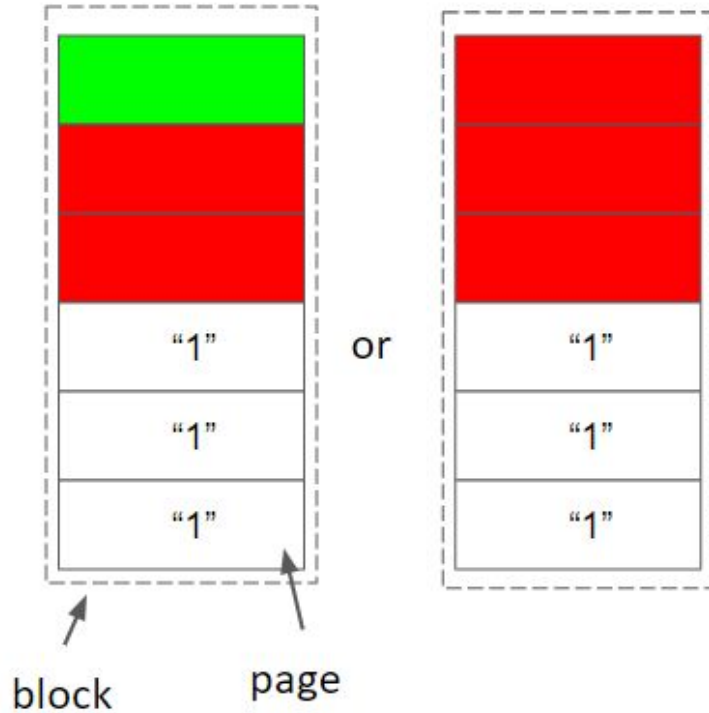


- A few pages of the block may be occupied by the hidden data and **cannot be decrypted**





Attack 1 (cont.)

With hidden volume



- Random data can **not** be decrypted to meaningful public data

 decryptable
 un-decryptable

Attack 2

DEFY: A Deniable, Encrypted flash-based File System which is built in the flash memory [Peters et al., 2015]

- A flash file system which supports PDE
- Multiple security levels, lower security level will not have any knowledge on the existence of the higher security levels
 - Data from the lower level may overwrite the data in higher level
 - Disables garbage collection in lower level
- Suffers from fill-to-full attack
 - Copy the device data elsewhere, deletes them from device
 - Writes the data back to the device
 - No more data are allowed to be written to the device, even though there is still a large amount of empty space

Adversarial Model

- The adversary will know the design of DEFTL. However, he/she does not have any knowledge on the keys and passwords of the PDE mode
- The adversary will stop coercing the device's owner once he/she is convinced that the decryption keys have been revealed
- The adversary cannot capture a device working in the PDE mode or after a crash of the PDE mode. Otherwise, he/she can trivially retrieve the sensitive data or detect the existence of PDE
- The operating system, bootloader, baseband OS, and firmware are all malware-free

Design of DEFTL

Overview

- How to prevent the sensitive data from being leaked to a coercive adversary ?
 - Adopt hidden volume technique
 - Modify block allocation, unmounting strategy, etc.
- How to prevent the hidden sensitive data from being overwritten by the non-sensitive data?
 - Isolate the public volume and hidden volume
 - Modify garbage collection & block allocation

Four Block Types:

A: Do not store any valid public or hidden data

B: Do not store any valid public data, but store valid hidden data

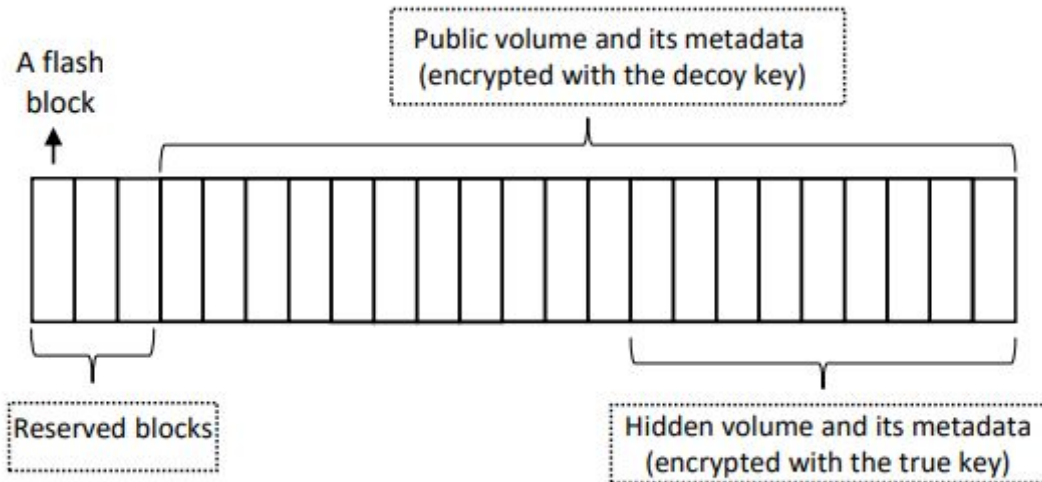
C: Contain both the valid public volume pages and the invalid public volume pages

D: Only contain valid public volume pages

Dirty Block Table: Stores the count of valid pages for each flash block, and organizes the blocks according to their counts in an increasing order

Initialization

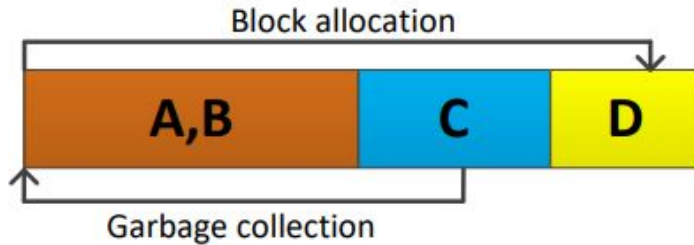
- Filling the entire flash with randomness
- Initializing the public and the hidden volume
 - Root table, bad block table and erasure count table will be stored in the reserved blocks



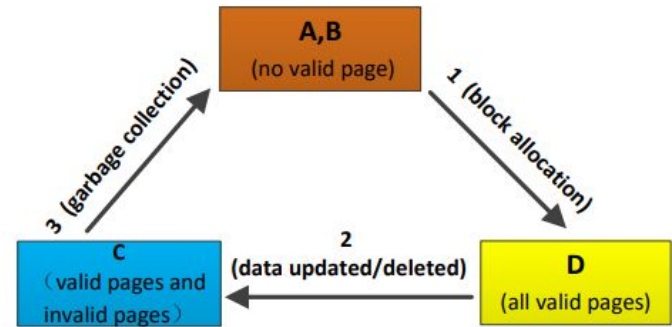
Public Mode

Block Allocation:

- Select the free blocks from the **head** of the dirty block table when a new write request comes
- Smartly manipulating the dirty block table of the public volume to ensure that it is more likely the blocks in state A will be allocated, rather than the blocks in state B



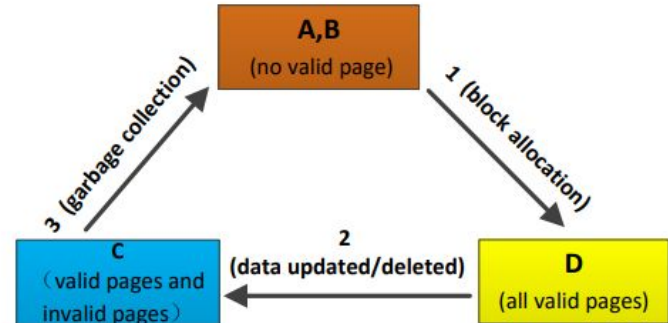
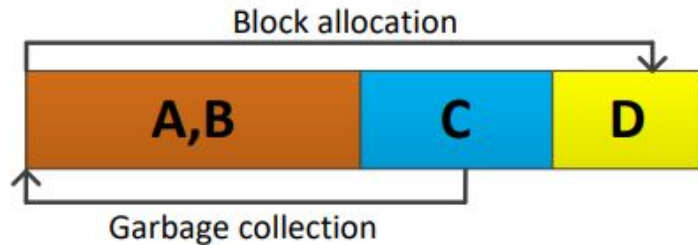
Dirty Block Table



Public Mode (cont.)

Garbage Collection:

- Perform active garbage collection over blocks in state C
- Reclaim blocks in state C when threshold is reached and relocate them to the **head** of dirty block table



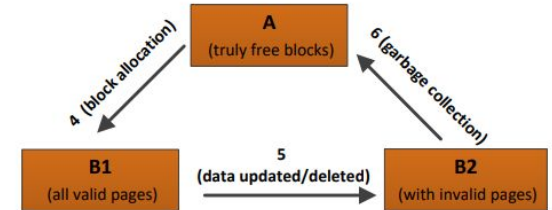
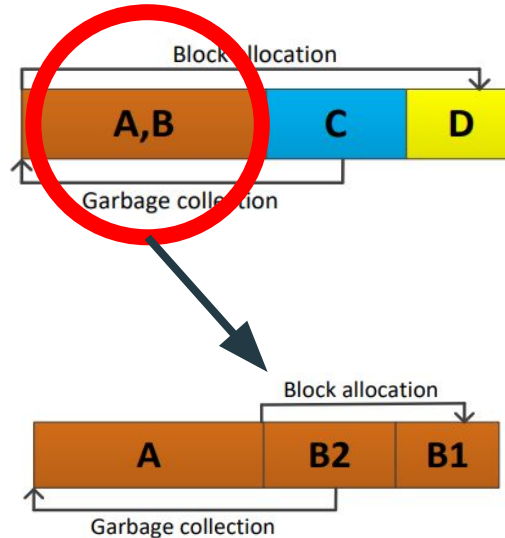
PDE Mode

Block Allocation:

- Select free blocks from the dirty block table from the **tail** of the blocks in state A

B1: The blocks which only contain valid hidden data

B2: The blocks which contain both valid and invalid hidden data



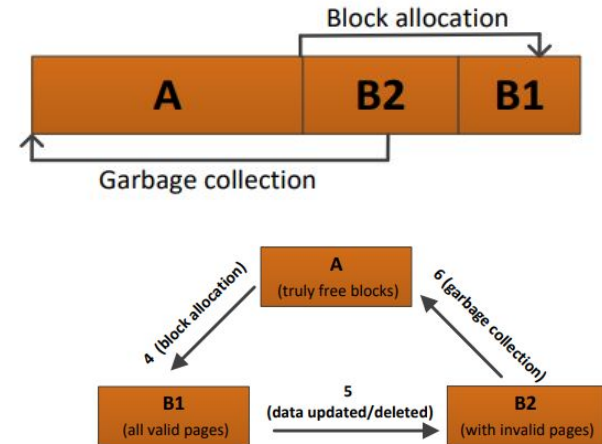
PDE Mode (cont.)

Garbage Collection:

- Perform active garbage collection over blocks in state B2
- Reclaim blocks in state B2 when threshold is reached and relocate them to the **head** of dirty block table

B1: The blocks which only contain valid hidden data

B2: The blocks which contain both valid and invalid hidden data



PDE Mode (cont.)

- **Bad block management:**
 - Erase the bad block immediately after reclaimed
- **Unmounting the hidden volume:**
 - Fill the blocks which have empty pages with randomness
 - Adjust the public volume dirty block table to be the same with the table of hidden volume

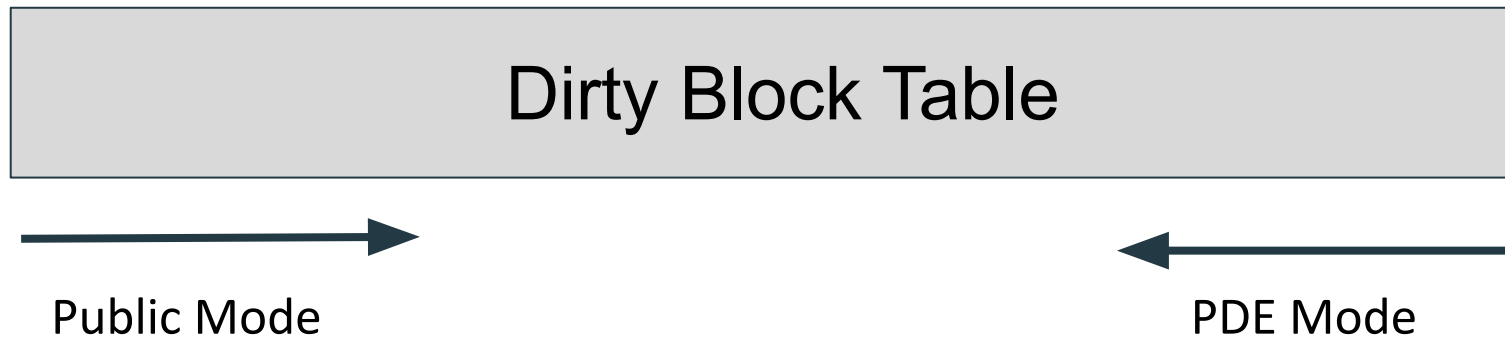
User Steps

Enter decoy password: Using the decoy password, DEFTL can derive the decoy key and use the decoy key to decrypt the public volume metadata

Enter true password: Using the true password, DEFTL can derive the true key and further localizes the hidden volume metadata, and decrypts them using the true key

Core Idea

- Public mode allocates the blocks from the **head** of dirty block table; PDE mode allocates the blocks from the **tail** of the dirty block table
- Garbage collection is performed **actively** both in public mode and hidden mode to make sure there are enough blocks to be used

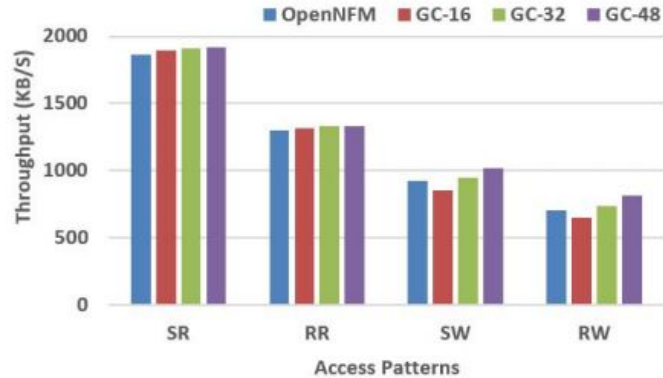


Implementation

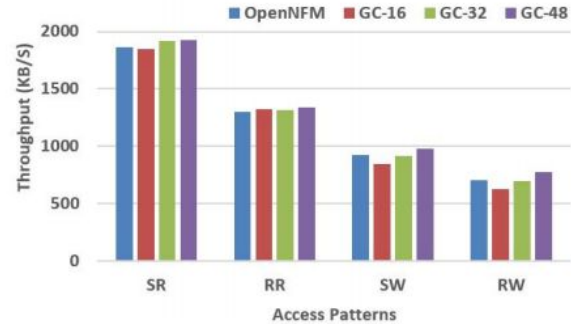
- Implement a prototype of DEFTL using OpenNFM
 - Two modes: public mode and hidden mode
- Port DEFTL to LPC-H3131
 - 180MHz ARM microcontroller
 - 512MB NAND flash

Evaluation

Throughput:



OpenNFM vs. Public Mode



OpenNFM vs. PDE Mode

Evaluation (cont.)

Wear Leveling:

Wear Leveling Inequality (WLI): Calculating an appropriately normalized sum of the difference of each measurement to the mean. Small WLI indicates a better wear leveling performance.

Wear leveling threshold	Average erasures	WLI (%)
200	0.97	11.5
150	1.06	10.2
100	1.10	8.9
50	1.15	7.3

Thoughts

- Only analyze the potential attacks theoretically
- Can **not** defend against a multiple-snapshot adversary
- Not user-friendly

Questions