Ensuring Data Confidentiality in Mobile Computing Devices via Plausibly Deniable Encryption and Secure Deletion

Presenter: Niusen Chen

Mobile Devices are Ubiquitous

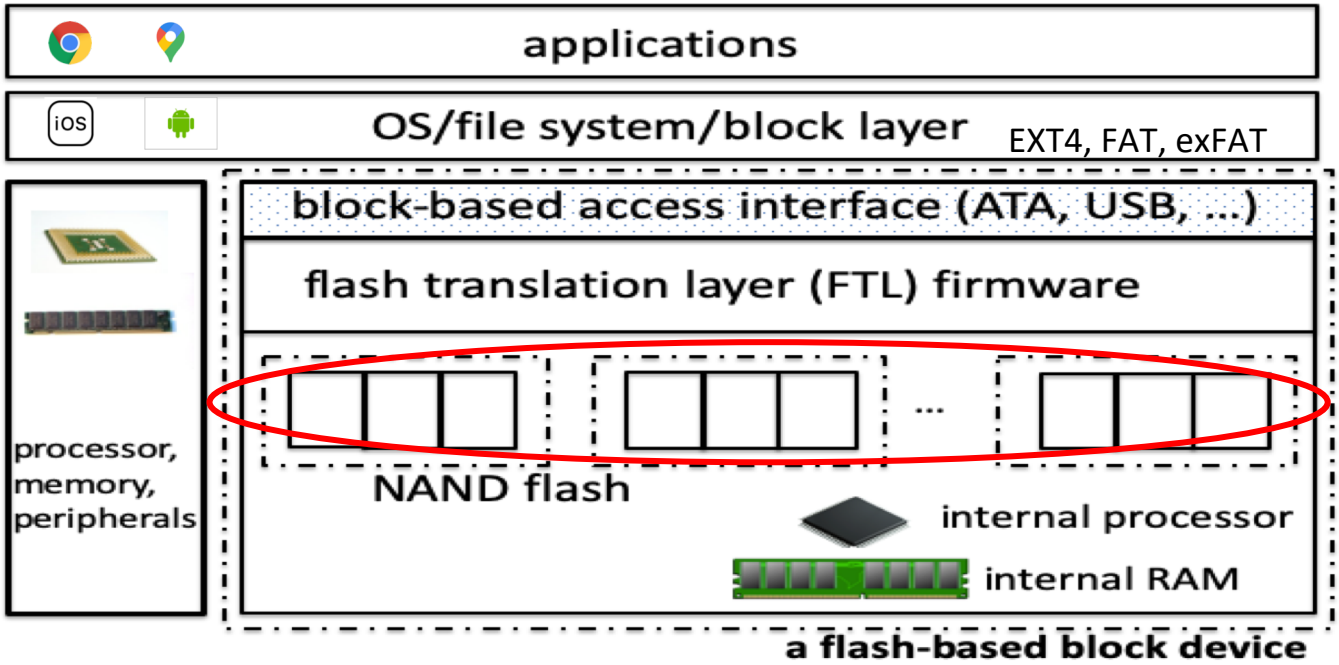
Smartphone

Tablet

Smartwatch (wearable device)

IoT devices

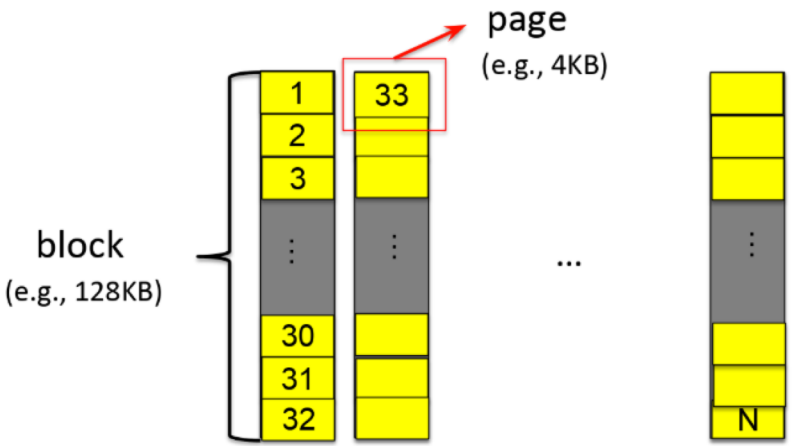
The Mainstream Architecture of Mobile Devices



Flash memory-based block device: A storage device based on high-speed, electrically programmable flash memory that supports reading and writing data in fixed-size blocks, sectors, or clusters. These blocks are generally 512 bytes

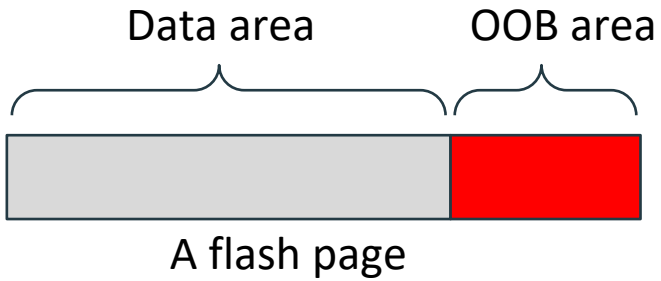
Hardware Characteristics of Flash Memory

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



Hardware Characteristics of Flash Memory (cont.)

5. Out-of-band (OOB) area



- Located at the end of each page
- Store extra information like error correcting code

6. SLC and MLC

a. SLC: each memory cell stores 1 bit
b. MLC: each memory cell stores more than 1 bit

Special Functions Incorporated into Flash Storage Device

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 reused

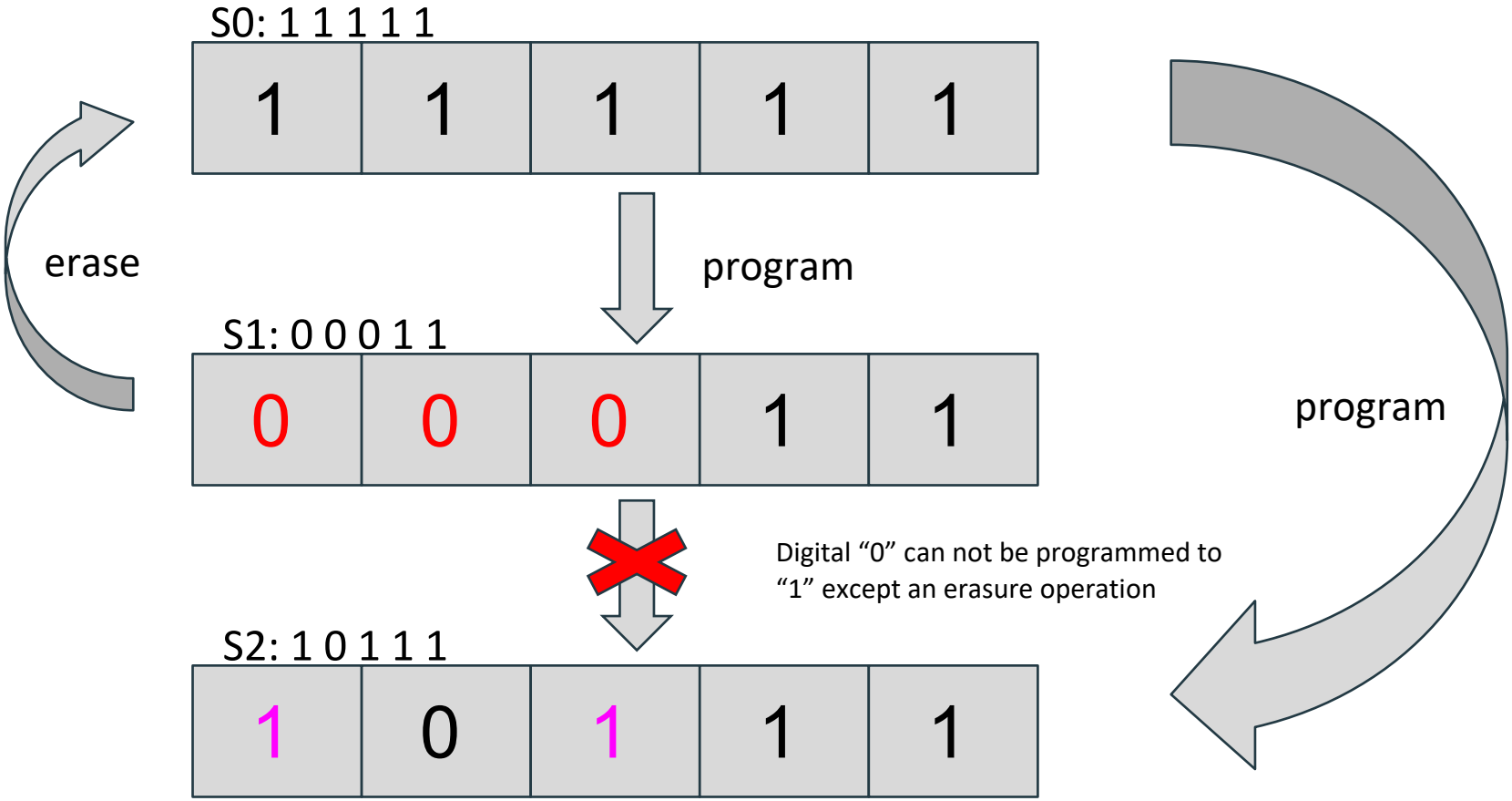
Wear Levelling: Distribute writes/erasures evenly across flash memory

Bad Block Management: A flash block may turn “bad” over time and cannot reliably store data. Bad block management typically introduces a bad block table to keep track of bad blocks. Once a block turns bad, it will be added to the bad block table and will no longer be used

How to Program Data to Flash Memory

Three rules:

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


S0: 1 1 1 1 1
1 1 1 1 1
erase
program
S1: 0 0 0 1 1
0 0 0 1 1
program
Digital “0” can not be programmed to “1” except an erasure operation
S2: 1 0 1 1 1
1 0 1 1 1

How to Use Flash Memory

File System (FAT, EXT4)

Flash Translation Layer
(FTL)

Flash Memory

Method 1: FTL

Flash-specific File System
(YAFFS, UBIFS)

Flash Memory

Method 2: Flash File System

Ensuring Data Confidentiality in Mobile Devices

How to ensure confidentiality of the sensitive data stored in mobile devices even when the mobile device owner is captured and coerced to disclose the key?

- Plausibly deniable encryption
- Ensure confidentiality of the data present in the storage media to defend against coercive attack (confidentiality during the data lifetime)

How to ensure that data deleted by a mobile device owner are really sanitized from the device?

- Secure deletion
- Ensure confidentiality of the data being deleted (confidentiality after the data lifetime)

Publications

[1] **Niusen Chen**, and Bo Chen. Duplicates also Matter! Towards Secure Deletion on Flash-based Storage Media by Removing Duplicates, under submission.

[2] **Niusen Chen**, Bo Chen, Weisong Shi. A Full-path Plausibly Deniable Encryption System for Mobile Devices, being submitted.

[3] **Niusen Chen**, Wen Xie, and Bo Chen. Combating the OS-level Malware in Mobile Devices by Leveraging Isolation and Steganography. *The Second ACNS Workshop on Secure Cryptographic Implementation (SCI '21)(in conjunction with ACNS '21)*, Kamakura, Japan, June 2021.

[4] **Niusen Chen**, Bo Chen, and Weisong Shi. MobiWear: A Plausibly Deniable Encryption System for Wearable Mobile Devices. *The First EAI International Conference on Applied Cryptography in Computer and Communications (AC3 '21)*, Xiamen, China, May 2021. (Best Paper Award, top 2.4% of all the submissions)

[5] Bo Chen, and **Niusen Chen**. **Poster**: A Secure Plausibly Deniable System for Mobile Devices against Multi-snapshot Adversaries. *2020 IEEE Symposium on Security and Privacy (S&P '20)*, poster paper, San Francisco, CA, May 2020.

[6] Wen Xie, **Niusen Chen**, and Bo Chen. **Poster**: Incorporating Malware Detection into The Flash Translation Layer. *2020 IEEE Symposium on Security and Privacy (S&P '20)*, poster paper, San Francisco, CA, May 2020.

Outline

- A Full-path Plausibly Deniable Encryption System for Mobile Devices (**MobiPDE**)
- A Plausibly Deniable Encryption System for Wearable Mobile Devices (**MobiWear**)
- A Secure Deletion Scheme for Flash-based Devices by Removing Duplicates (**RedFlash**)

A Full-path Plausibly Deniable Encryption System for Mobile Devices (MobiPDE)

How to Protect the Confidentiality of Existing Data

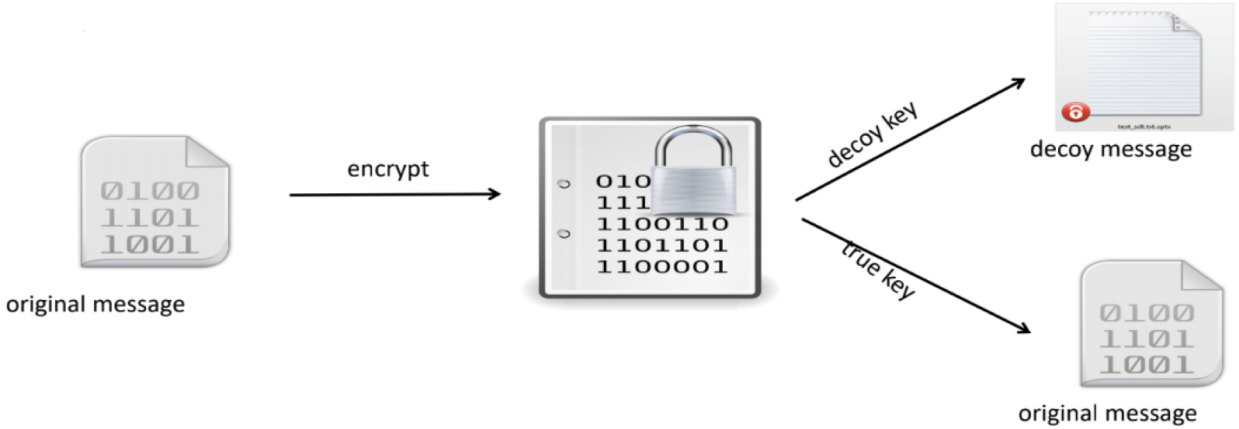
- Full Disk Encryption (FDE)
  - Everything on disk is encrypted
  - Totally transparent to users
  - Cannot defend against a coercive attacker
  - Examples: VeraCrypt, TrueCrypt, BitLocker

Coercive Attacks

An attacker forces the device owner to disclose the decryption key

Plausibly Deniable Encryption (PDE)

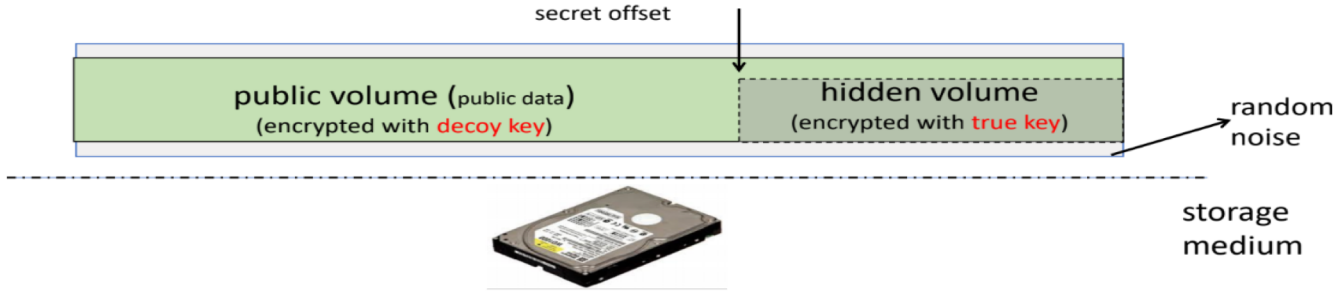
- A crypto primitive designed for mitigating coercive attacks
- Plaintext is encrypted by a true key and a decoy key such that:
  - Decrypted with decoy key → Decoy message
  - Decrypted with true key → True (original) message
- Upon being coerced: disclose decoy key, keep true key
- Existence of hidden sensitive data can be denied reasonably (plausible deniability)



A Typical Implementation of PDE

Hidden Volume Technique

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



Existing PDE Works for Mobile Devices

	PDE technique	storage layer	secure	user-oriented	Compatible with mainstream mobile devices
MobiFlage[Skillen et al., 2013]	hidden volume	block device	no	yes	yes
MobiHydra[Yu et al., 2014]	hidden volume	block device	no	yes	yes
MobiPluto[Chang et al., 2018]	hidden volume	block device	no	yes	yes
DEFY[Peters et al., 2015]	steganography	flash file system	partially	yes	no
DEFTL[Jia et al., 2017]	hidden volume	FTL	yes	no	yes
MobiMimosa[Hong et al., 2017]	hidden volume	block device	no	yes	yes
MobiCeal[Chang et al., 2018]	steganography	block device	no	yes	yes
MobiGyges[Feng et al., 2020]	hidden volume	block device	no	yes	yes
INFUSE[Chen et al., 2020]	side channel	flash file system	yes	yes	no
PEARL[Chen et al., 2021]	WOM codes	FTL	yes	no	yes

Conclusion

- Embedding PDE in block device layer is user-oriented but not secure
  - Unique nature and special functions (e.g., garbage collection and wear leveling) of flash memory may compromise the deniability
- Embedding PDE in FTL layer is secure but not user-oriented
  - User cannot access to FTL layer
- Designing a file system supporting PDE is not compatible with the architecture of mainstream mobile devices
  - Most of the mainstream mobile devices use FTL to manage flash memory

A **full-path** mobile PDE system which is secure, being compatible with the storage architecture of mainstream mobile devices, lightweight as well as user-oriented is missing

We aim to build a full-path mobile PDE system

Design Rationale

Technique: hidden volume technique

- Public volume
- Hidden volume

Two modes in user level:

- Public mode: manage public volume
  - Read and write public non-sensitive data
- Hidden mode: manage hidden volume
  - Read and write hidden sensitive data

Q1: Where can we deploy the public/ hidden volume?

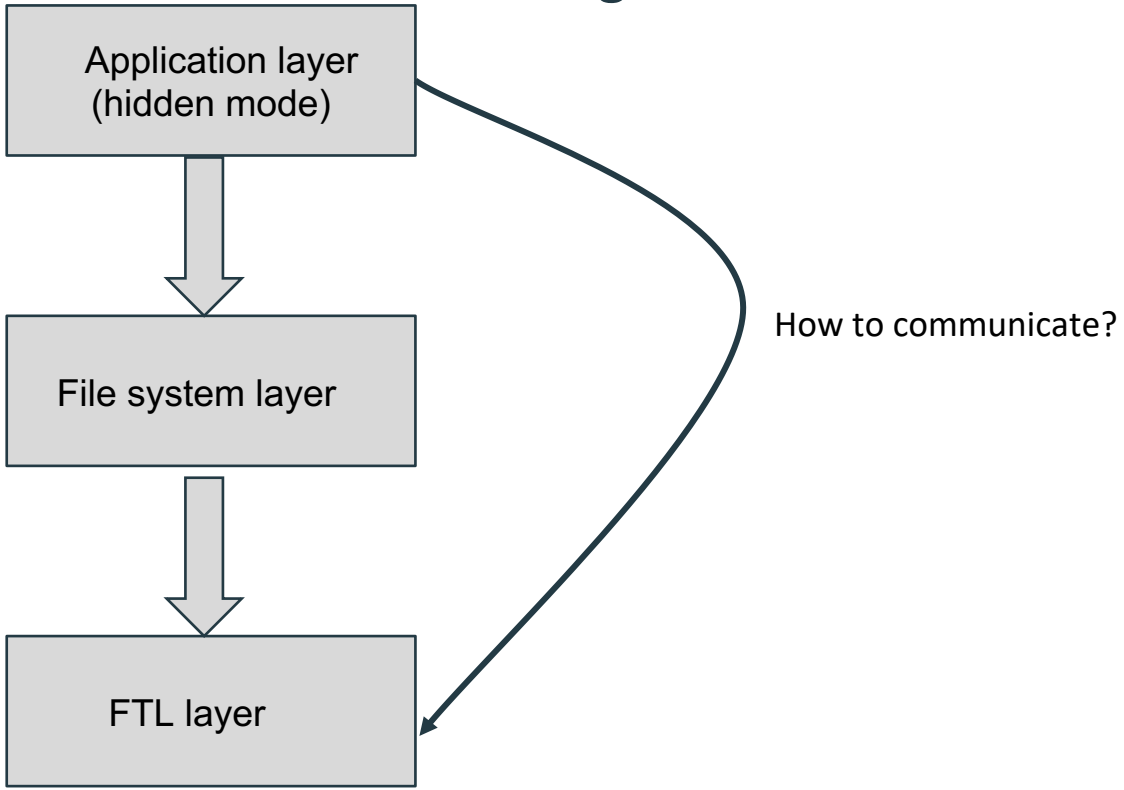
Deploy public and hidden volume to the block layer:

- OS can easily manage both volumes using hardware of the host computing device
- The host computing device's hardware is more powerful than the internal hardware of the flash-based block device
- Easily deploy a file system on top of the public volume and the hidden volume

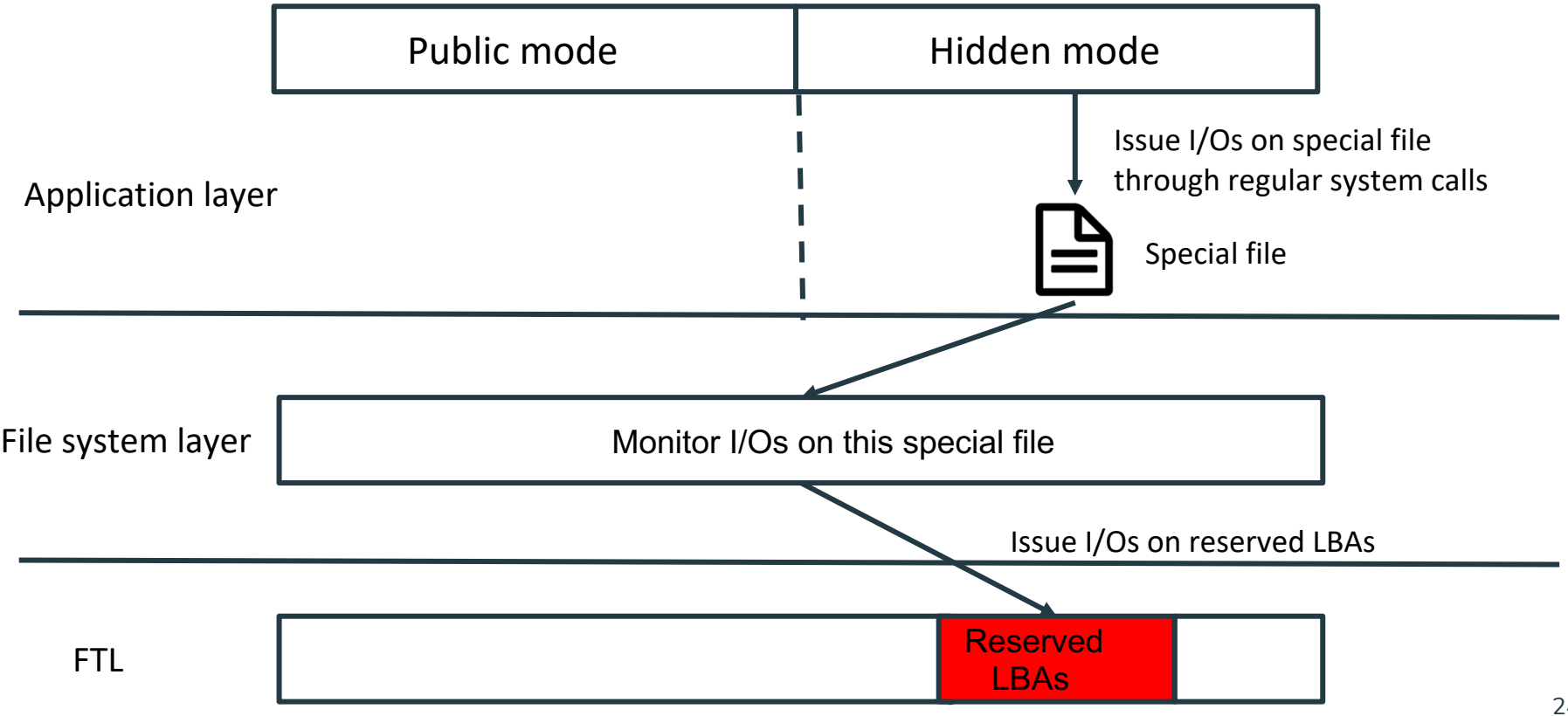
Q2: How can we mitigate loss of hidden sensitive data?

- Block layer
  - Hide sensitive data at the end of the disk
  - Use a file system which writes data sequentially from the beginning of the disk (e.g., FAT, exFAT)
- FTL layer
  - There is no data loss in FTL layer since FTL will not use the pages occupied by the hidden data
- User
  - Should be careful when using the public volume
  - Should know the existence of hidden volume and pay attention to the disk space used

Q3: How can the hidden mode securely communicate with the FTL to control those flash blocks storing hidden data?



Q3: How can the hidden mode securely communicate with the FTL to control those flash blocks storing hidden data?



Q3: How can the hidden mode securely communicate with the FTL to control those flash blocks storing hidden data?

Advantages

- Maintaining the existing system calls
- Maintaining the existing I/O interfaces of the block device

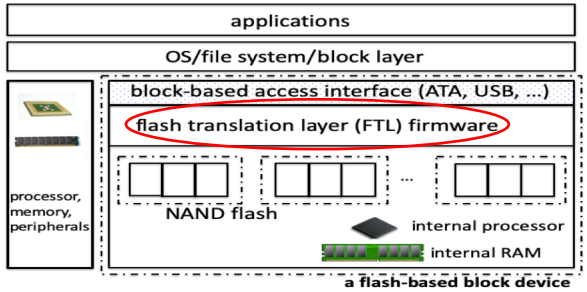
Q4: How can we avoid the deniability compromises when an adversary can have access to the raw NAND flash?

FTL incorporates separate logic to manage hidden sensitive data to avoid deniability from being compromised

- Maintain a separate data structure to keep track of pages invalidated by the hidden mode in FTL, this data structure is invisible to the public mode
- FTL will not move flash blocks to the free block pool in hidden mode
- FTL has separate functions (e.g., garbage collection, wear leveling) for hidden mode
- When writing a flash page in the hidden mode, we will always commit its corresponding logical address of the public mode (rather than that of the hidden mode) to its OOB

Design Summary

Flash Translation Layer (mitigate deniability compromise):



- Block allocation
  - Write from the beginning in public volume
  - Write from the end in hidden volume
  - Do not update OOB area of each flash page occupied by hidden data
- Garbage collection & Wear leveling
  - Public mode: follow the original FTL
  - Hidden mode: use separate garbage collection and wear leveling
- Other operations
  - Monitor I/Os from upper layer to securely communicate with hidden mode

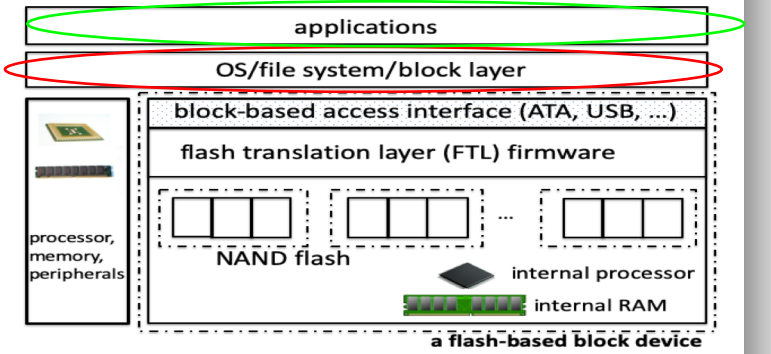
Design Summary (cont.)

Block Layer (improve performance, prevent data loss):

- Both the public and the hidden volume are deployed on the block layer
- Encryption/decryption are conducted in block layer
- Host computing device is more powerful than flash-based device

File System Layer:

- Monitor I/Os from application layer
- Issue I/Os on a few reserved logical addresses to securely communicate with FTL layer



Application Layer:

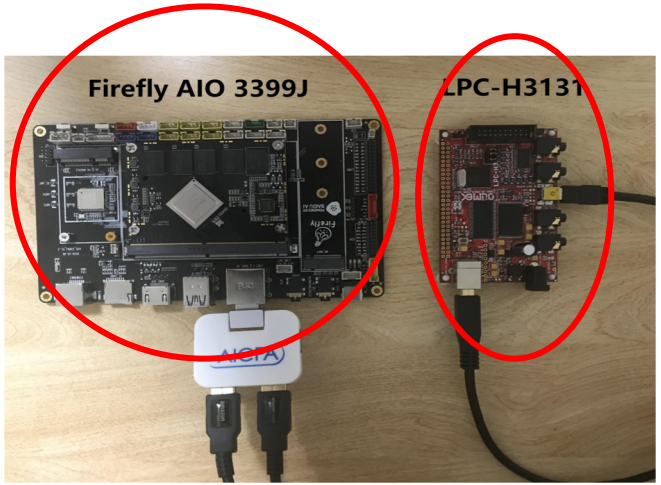
- Send different requests by issuing different I/O patterns on special file

Implementation

- FTL: Modified OpenNFM[1]
  - Block allocation
  - Garbage collection and wear leveling

- File system layer: Modified exFAT
  - Monitor I/Os from application layer
  - Issue I/Os on a few reserved logical addresses to securely communicate with FTL layer

- Application layer: Modified Veracrypt
  - Send different requests by issuing different I/O patterns on special file



[1]Google Code. Opennfm. https://code.google.com/p/opennfm/, 2011

Evaluation (cont.)

patterns	Public mode (KB/s)	Hidden mode (KB/s)
Sequential read	2508	2473
Random read	2174	2030
Sequential write	2599	2372
Random write	1897	1842

Throughput of Veracrypt

patterns	Public mode (KB/s)	Hidden mode (KB/s)
Sequential read	2460	2424
Random read	2086	2000
Sequential write	2535	948
Random write	1910	839

Throughput of MobiPDE

Drawbacks of MobiPDE:

- MobiPDE relies on a strong assumption that filling randomness is a normal system behavior
- Wearable devices are popular, but MobiPDE is not a perfect design for wearable devices
  - Filling randomness is a very expensive operation for wearable devices

Can we design a PDE system for wearable devices such that:

- Do not rely on the strong assumption that filling randomness is a normal system behavior
- Match the hardware characteristics of wearable devices
  - Small size
  - Small screen
  - No keyboard

A Plausibly Deniable Encryption System for Wearable Mobile Devices (MobiWear)

Image Steganography

- It is used to hide information in a cover image
- It can be performed in two domains:
  - Spatial domain
  - Transform domain (more computationally intensive)

Spatial Domain

The least significant bit (LSB) steganographic embedding technique

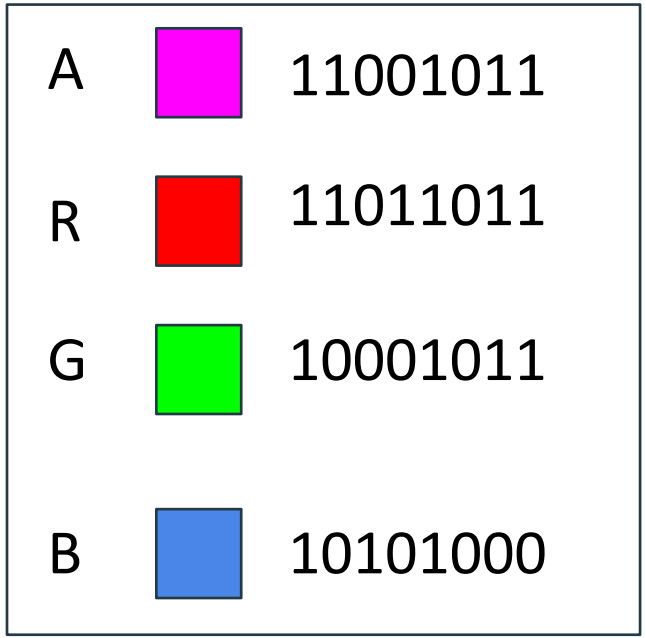
- Each pixel can be represented by 4 bytes in an ARGB image
  - R: red
  - G: green
  - B: blue
  - A: transparency

- Secret data are hidden in the least significant bit(s) of each pixel

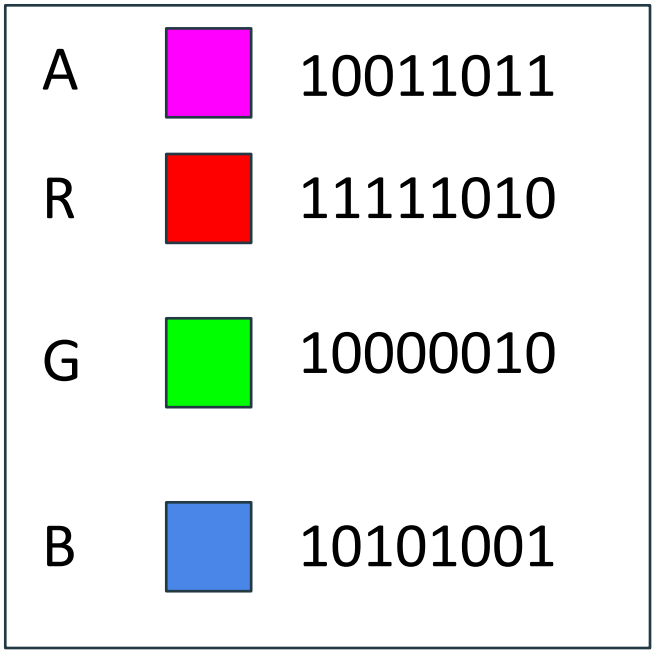
- The LSB technique has minimal effects on the image quality and is lightweight

Spatial Domain (cont.)

“A”: 0 1 0 0 0 0 0 1

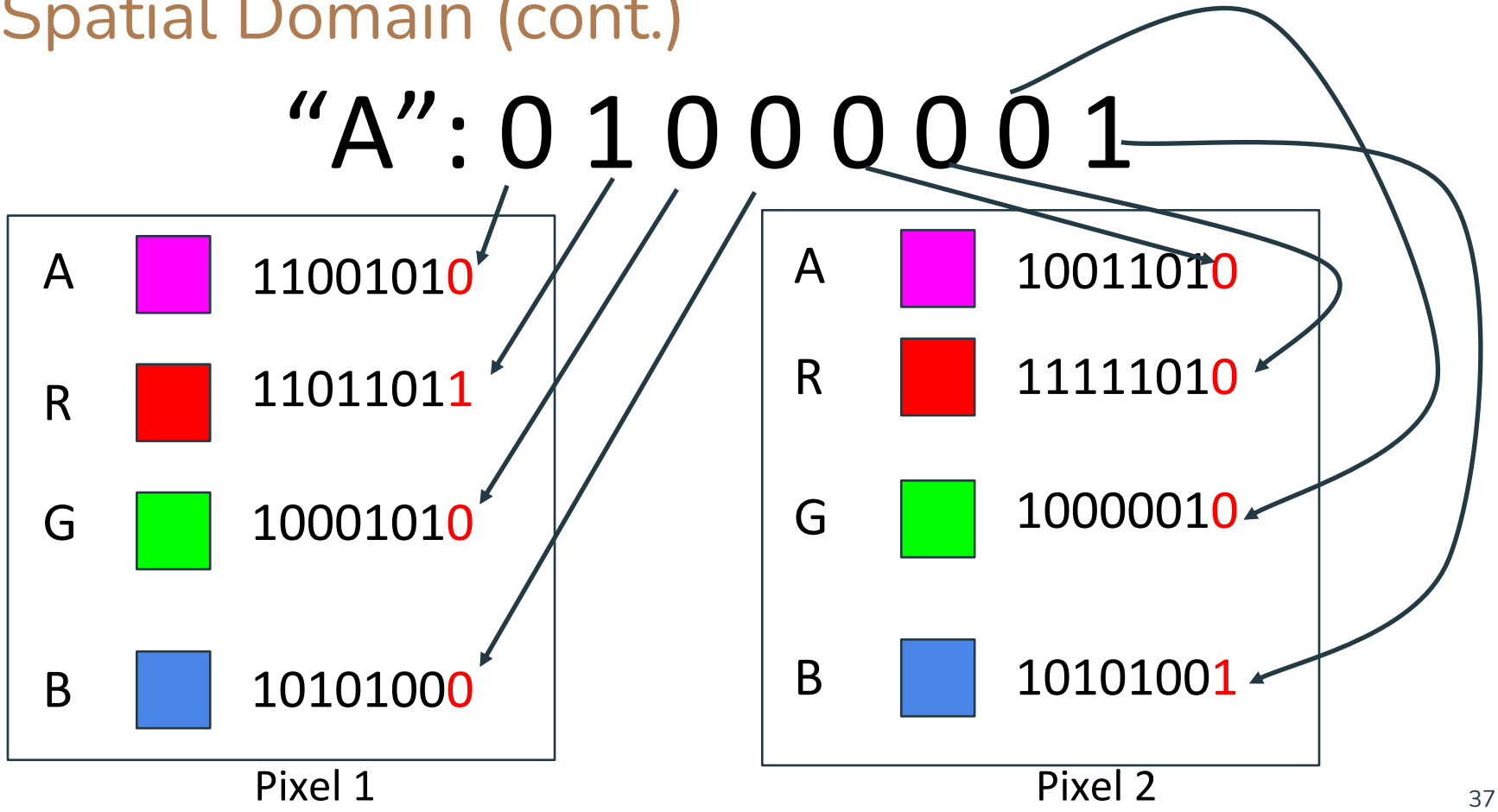


Pixel 1



Pixel 2

Spatial Domain (cont.)



Digital Watermarking

- It is typically used to identify ownership of the copyright
- It includes visible watermarking and invisible watermarking

Original Image

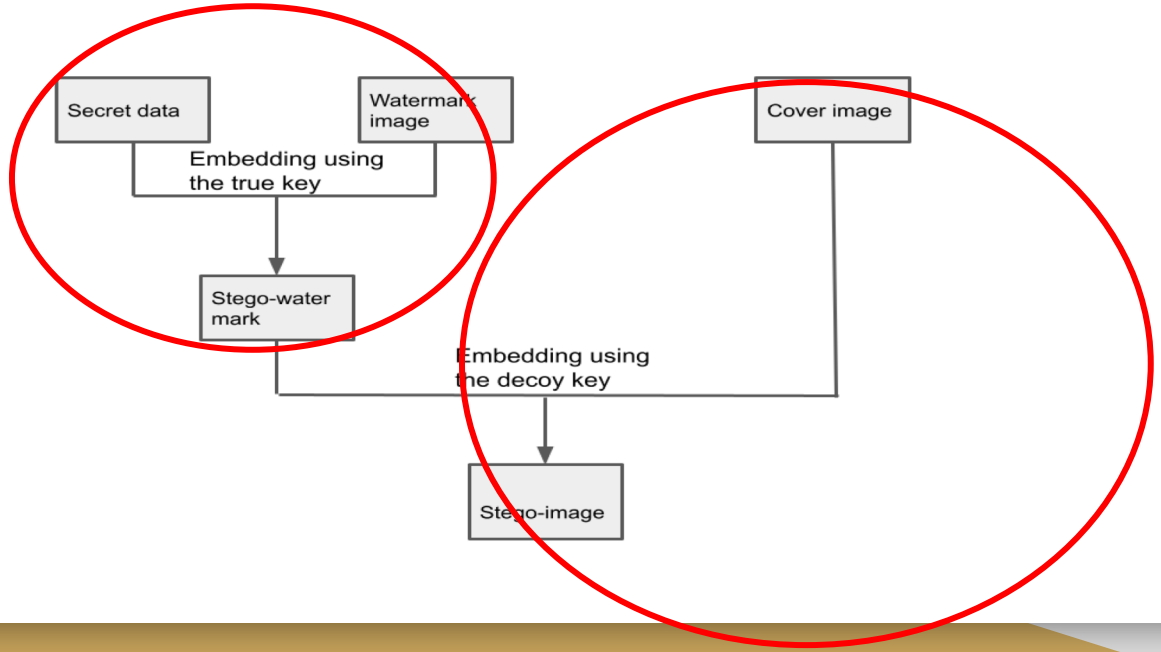
Encoded Image (watermark)

Key Insights

- Combine image steganography and watermarking to implement PDE
  - Do not use the hidden volume technique, which 1) requires filling randomness to the device initially; 2) relies on the assumption that filling randomness is a normal system behavior
- The sensitive data are denied as the watermarks embedded into the images
  - Upon being coerced, the victim will disclose the decoy key and claim that there is a watermark (rather than hidden sensitive data) embedded in the cover image
- Rely on embedded sensors to input keys
  - Match the hardware characteristics of wearable devices

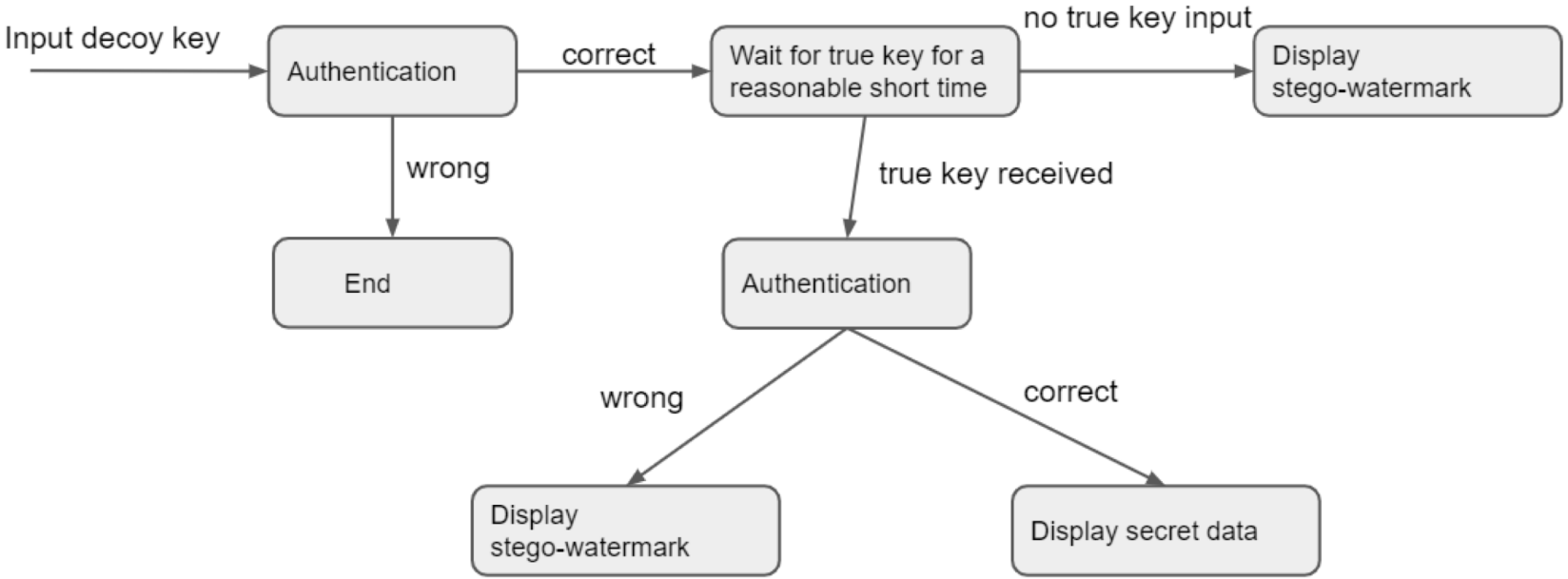
Our Design

- Secret data are encrypted with true key and hidden in watermark → stego-watermark
- Stego-watermark is encrypted with decoy key and hidden in cover image → stego-image



Our Design (cont.)

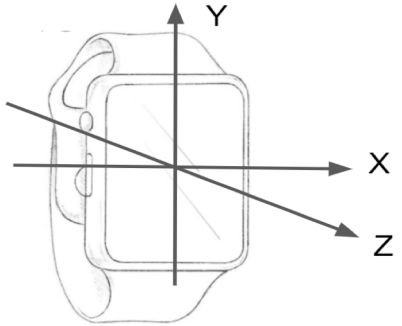
User authentication:



Our Design (cont.)

How to input keys in wearable device:

- Due to the small size of a wearable mobile device, using a keyboard or a touchscreen to input keys is inconvenient
- We use gyroscope sensor to input the keys
  - Convenient for users to enter the keys (e.g., rotate the wrist)
- How to input keys via gyroscope sensor
  - User rotates the wrist in different directions to enter keys
  - Gyroscope will measure the rotation rate and calculate the rotation degree in a time interval of each direction
  - A key can be generated by combining degrees of x, y , and z-axis
  - Set a threshold when comparing the keys



Evaluation

- Implement MobiWear in an LG G watch
  - 512MB RAM, 4GB storage
  - OS: Android Wear 1.5
- Evaluate whether a user can authenticate successfully via gyroscope under different thresholds
- Evaluate Peak Signal-to-Noise-Ratio (PSNR) to indicate image quality
- Evaluate processing time for hiding/extracting sensitive data

Evaluation(cont.)

	5.5 (5<= x < 6)	25.5 (25 <= x < 26)	45.5 (45 <= x < 46)	65.5 (65 <= x < 66)
Gyroscope-X	√	√	√	√
Gyroscope-Y	√	√	√	√
Gyroscope-Z	√	√	√	√

Threshold = 0.5°

	5 (4<= x < 6)	25 (24 <= x < 26)	45 (44 <= x < 46)	65 (64 <= x < 66)
Gyroscope-X	√	√	√	√
Gyroscope-Y	√	√	√	√
Gyroscope-Z	√	√	√	√

Threshold = 1°

Evaluation(cont.)

	5 (3<= x < 7)	25 (23 <= x < 27)	45 (43 <= x < 47)	65 (63 <= x < 67)
Gyroscope-X	√	√	√	√
Gyroscope-Y	√	√	√	√
Gyroscope-Z	√	√	√	√

Threshold = 2°

Evaluation (cont.)

PSNR: Represents the ratio between the maximum possible power of a signal and the power of the noise. Higher PSNR value indicates good image quality

$$PSNR = 20 \cdot log_{10}(MAX_I) - 10 \cdot log_{10}(MSE)$$

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2$$

m,n: the size of the original image is mxn
I: original image
K: noisy approximation of original image
MAX_I: maximum pixel value of the image

Evaluation (cont.)

Length of secret data (bytes)	20	40	60
PSNR	32.0192	31.9214	31.9048

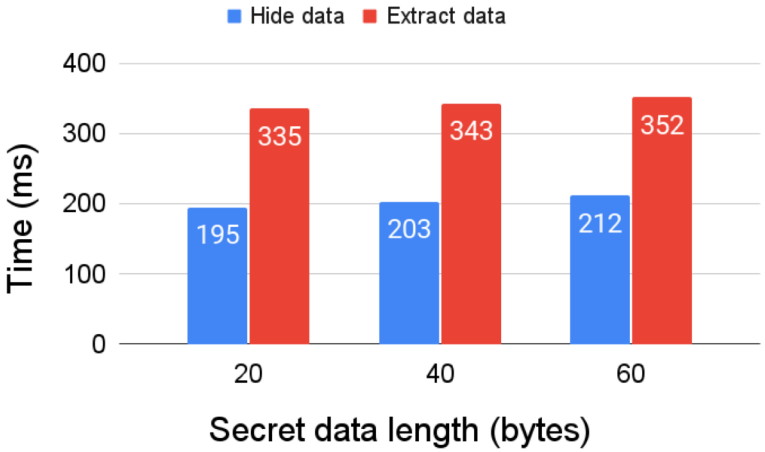
PSNR of stego-images under different lengths of secret data

Length of secret data (bytes)	20	40	60
PSNR	34.3125	34.2577	33.2385

PSNR of stego-watermarks under different lengths of secret data

Conclusion: Longer secret data will result in lower PSNR values

Evaluation (cont.)

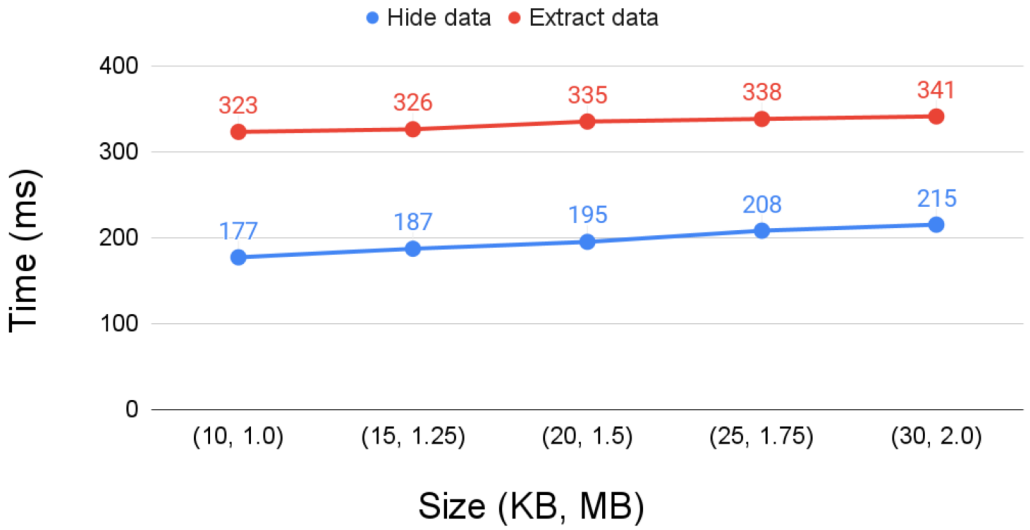


Processing time with different secret data lengths

Conclusion:

1. Longer secret data needs more time in hiding/ extracting secret data
2. Extracting the secret data is slower than hiding them

Evaluation (cont.)



Processing time with different size of watermark/cover image

Conclusion:

1. The time for hiding/ extracting the secret data slightly increases when the size of the watermark/cover image increases
2. The time for extracting the secret data is more than that for hiding them

Discussion

- Length of sensitive data which can be hidden
  - Suppose cover image is **N** pixels, each pixel consists of 4 bytes (ARGB)
  - Maximal size of watermark: N/2 bytes
  - Maximal length of secret data: N/16 bytes
  - Using more LSB to hide sensitive data
- Deniability compromise in memory
  - Secret data may leave traces in memory
  - Power-off the device
  - Utilizing the hardware isolation technique (e.g., ARM TrustZone)

Discussion (cont.)

- Mitigating data corruptions
  - LSB technique is vulnerable to image cutting and cropping attack
  - Back up the data periodically
- Hiding data in other types of multimedia
  - Video and audio

Ensuring Data Confidentiality in Mobile Devices

How to ensure confidentiality of the sensitive data stored in mobile devices even when the mobile device owner is captured and coerced to disclose the key?

- Plausibly deniable encryption
- Ensure confidentiality of the data present in the storage media to defend against coercive attack (confidentiality during the data lifetime)

How to protect confidentiality after the data lifetime

- Secure deletion
- Ensure confidentiality of the data being deleted

Secure Deletion

Secure deletion guarantee

- Remove the data to be deleted
- The attacker cannot derive any sensitive information about the deleted data

Traditional Secure Deletion Methods are Not Sufficient for Mobile Devices

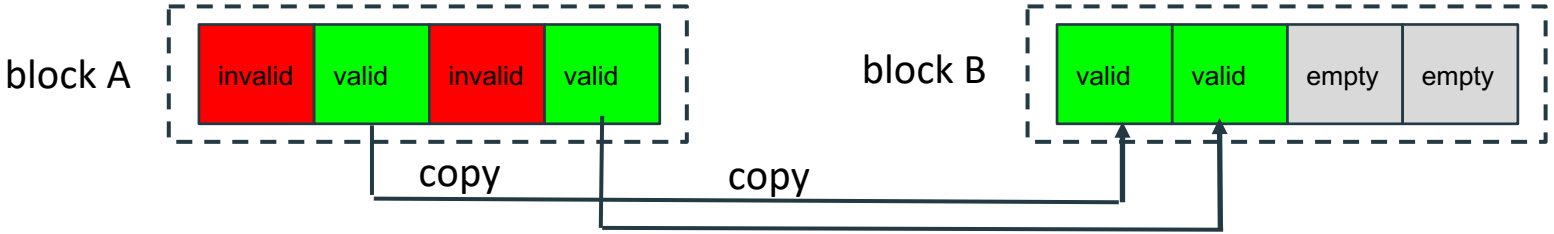
- Traditional secure deletion methods (overwrite or encryption) cannot achieve secure deletion guarantee in flash-based storage devices
  - Unique nature and special functions may cause various duplicates in flash memory, but traditional secure deletion methods do not handle them
  - Secure deletion guarantee may be compromised if attacker can have access to those duplicates

We aim to design a secure deletion scheme which can efficiently remove both data and the corresponding duplicates across the entire flash

A Secure Deletion Scheme for Flash-based Devices by Removing Duplicates (RedFlash)

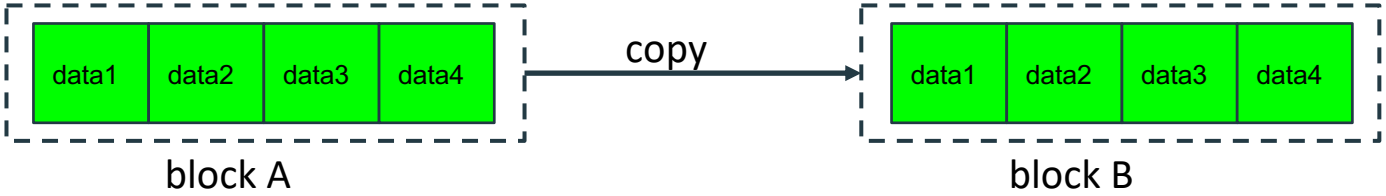
Theoretically Analyzing the Existence of Duplicates in Flash Storage Media

- Duplicates created by garbage collection



block A: victim block with the largest number of invalid pages
block B: free block

- Duplicates created by wear leveling

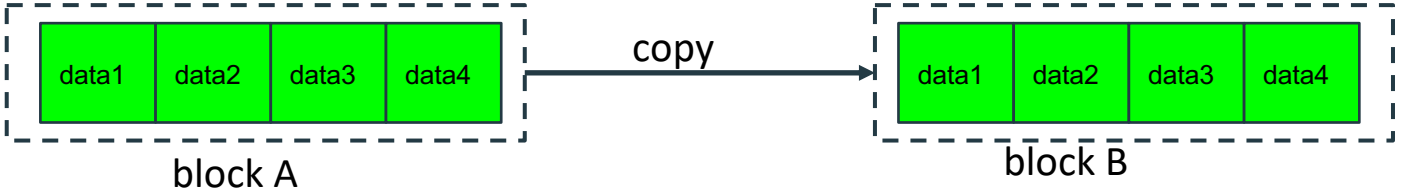


block A: a block which has the smallest erase count from blocks being used
block B: a block which has the largest erase count from the free block pool

Duplicates are not deleted immediately due to performance consideration

Theoretically Analyzing the Existence of Duplicates in Flash Storage Media (cont.)

- Duplicates created by bad block management



block A: bad block

block B: empty block from free block pool

Duplicates are not deleted immediately due to performance consideration

Experimentally Confirming the Existence of Duplicates

Experiment Setup:

- Host: Ubuntu 14
- Simulator: DiskSim 4.0 with SSD Model
- Trace Set: UMass Trace Repository
  - Financial : Financial1 + Financial2
  - WebSearch: WebSearch1 , WebSearch2 , WebSearch3

Trace Set Number	Content	Number of I/Os
1	Financial + WebSearch1	10,089,261
2	Financial + WebSearch2	13,613,622
3	Financial + WebSearch3	13,295,518

Trace Sets

Experimentally Confirming the Existence of Duplicates (cont.)

Experiment Process:

- We run three trace sets separately
- We calculate the total duplicate pages generated during each run
  - We only calculate the duplicate pages generated by wear leveling and garbage collection since the simulator cannot simulate bad block management

Experimentally Confirming the Existence of Duplicates (cont.)

Results:

Trace Set Number	Total Duplicate Pages Generated	Size of Duplicates (GB)
1	286,393	2.18
2	320,810	2.44
3	286,887	2.19

Simulation results

Conclusion:

- The total size of duplicate data generated during the whole process are approximately 2 GB
- Different traces will generate different number of duplicate pages. This is because different traces have different I/O operations, which will lead to different behaviors in the FTL layer, and eventually affect the number of duplicate pages generated

How to Securely Remove Data as well as the Corresponding Duplicates

Basic solution:

- We immediately delete any duplicates produced by internal management of flash storage (e.g., garbage collection, wear leveling, and bad block management)
- Once a deletion request is issued by the user from upper layer, we will immediately delete the corresponding data node

Disadvantages:

- For those data nodes which have duplicates but have not yet been deleted, it is unnecessary to remove their duplicates, as the existence of those duplicates does not “hurt” the secure deletion guarantee
- This solution is expensive and will decrease the performance

How to Securely Remove Data as well as the Corresponding Duplicates (cont.)

Solution:

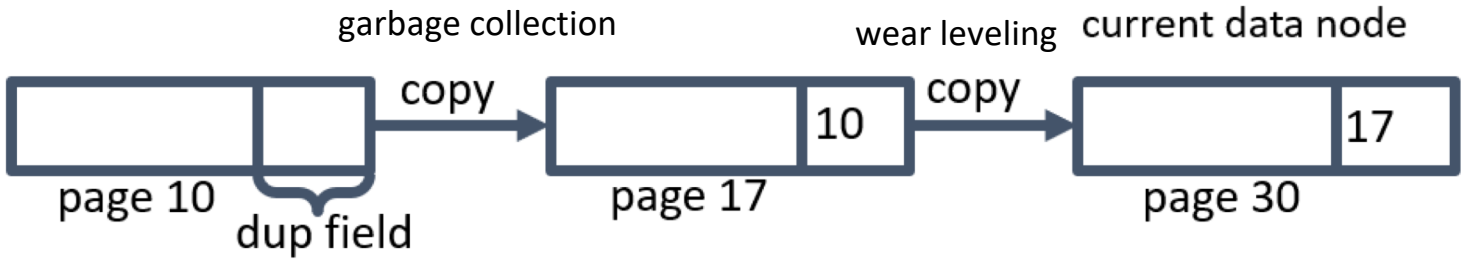
When a secure deletion request is issued, we first delete the targeted data node, and then locate and delete any duplicates across flash memory associated with this data node

- How to efficiently keep track of all duplicates?
- How to efficiently delete data?
  - Delete data in SLC flash
  - Delete data in MLC flash

How to Efficiently Keep Track of All Duplicates?

Our solution:

- We propose to “chain” each data node and its associated duplicates together in the flash memory, generating a duplicate chain



An example of duplicate chain

Advantages:

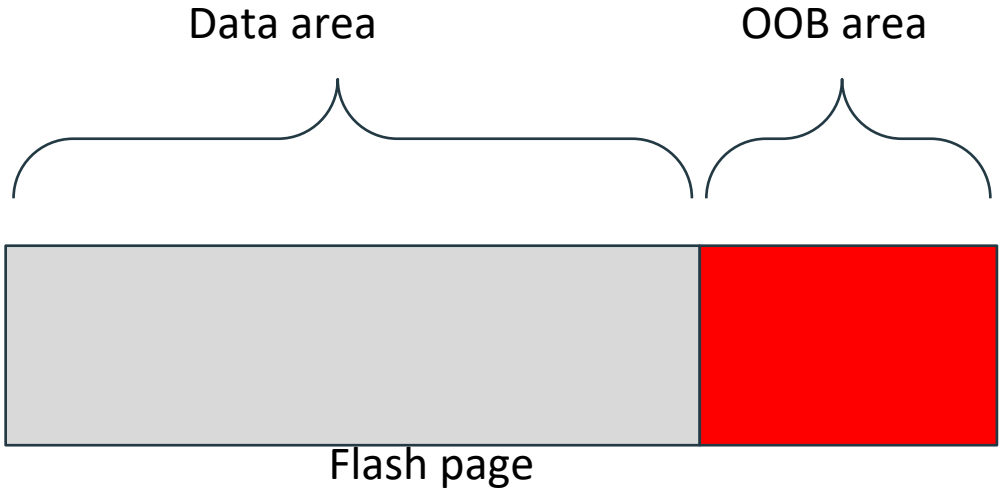
- **No searching**
  - Searching the duplicates across the entire flash device is expensive
  - May lead to mistakenly delete identical data belonging to another file
- **No RAM**
  - RAM can be used to track the locations of all duplicates associated with each data node
  - RAM is volatile and suffers from power-loss
  - The embedded flash device is usually equipped with a limited amount of RAM

How to Efficiently Keep Track of All Duplicates? (cont.)

Challenge 1: Where can we store the “dup field” in a flash page?

- Each flash page has an OOB area, and only a few bytes of OOB have been used by the flash controller

Mitigation:We utilize the remaining unused space of the OOB to store the “dup field”



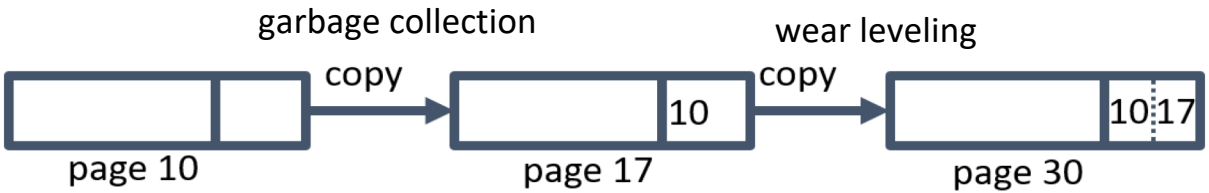
How to Efficiently Keep Track of All Duplicates? (cont.)

Challenge 2: How can we handle a “broken” duplicate chain?

- A duplicate in a chain may be reclaimed earlier and the entire chain could be broken
- Updating the chain is infeasible except performing an expensive block erasure

Mitigation: When adding a new page to the chain, we redundantly store all the prior locations in the chain to the “dup field” of this new page

An example of broken duplicate chain



A robust duplicate chain

How to Efficiently Keep Track of All Duplicates? (cont.)

Challenge 3: How can we handle the growth of “dup field”?

- The size of the “dup field” may grow over time
- The size of OOB area is limited

Mitigation:

- We periodically purged obsolete locations in “dup filed”
- In the worst case, we will use regular space from the page to store it

How to Securely Remove Data as well as the Corresponding Duplicates (cont.)

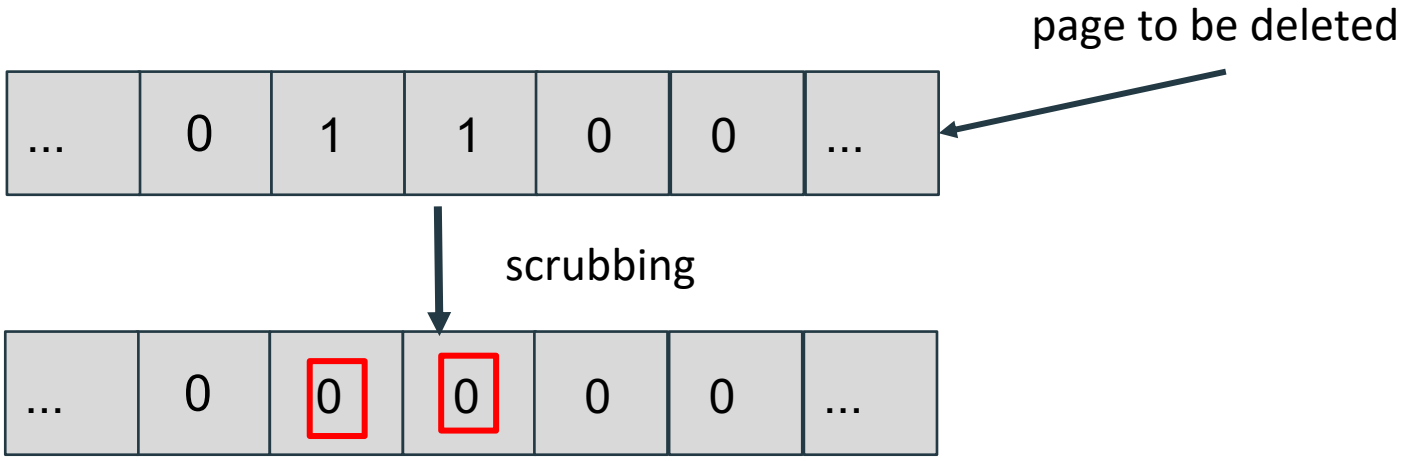
- How to efficiently keep track of all duplicates?
- How to efficiently delete data?
  - Delete data in SLC flash
  - Delete data in MLC flash

How to Efficiently Delete Data?

SLC:

Wei et al. (FAST ’11) proposed scrubbing to delete data from a flash page without erasing the corresponding block

- Flash allows programming ‘1’ bits to ‘0’ bits
- Programming all ‘1’ bits on a page to ‘0’ bits to delete data



How to Efficiently Delete Data? (cont.)

MLC:

- Simply scrubbing the target page cannot work
  - Scrubbing one page will cause corruptions to other pages since multiple bits share one cell in MLC

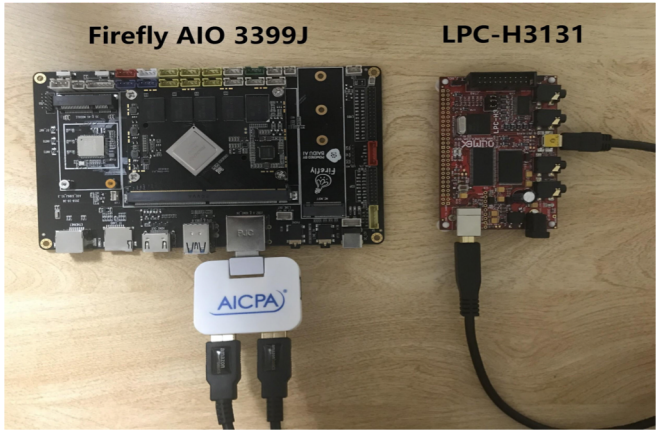
Solution:

- When secure deletion is triggered, for each target page in the duplicate chain, the content of its paired-page will be read and written to a new flash page
- Perform scrubbing on both the target and the paired page

Evaluation and Simulation

Real-world implementation and experimental setup:

- Host computing device: Firefly AIO 3399J
  - Six-Core ARM 64-bit processor
  - 4GB RAM
  - Kernel: 4.4.194
- Flash-based device: LPC-H3131
  - ARM9 -32bit ARM926EJ-S
  - 32MB RAM
  - 512MB NAND flash
- Implement RedFlash into OpenNFM



Evaluation and Simulation (cont.)

Evaluating lengths of duplicate chains:

Wear leveling threshold	Length
10	3
20	3

The length of the longest duplicate chains

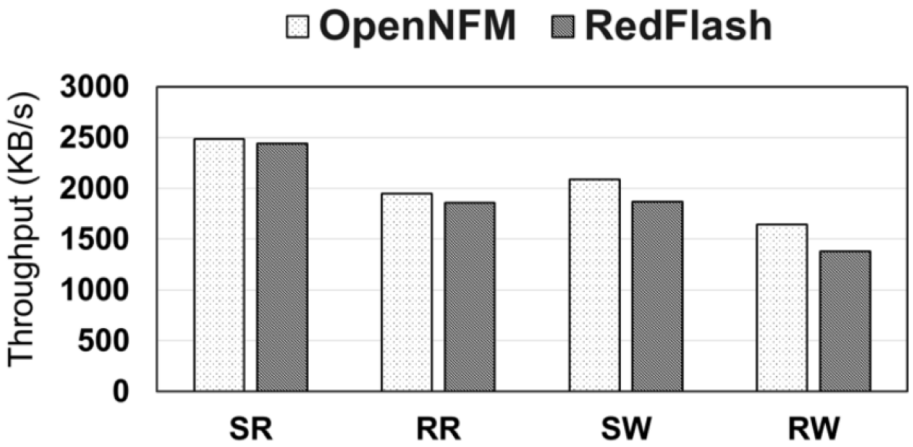
Conclusion:

We assess potential lengths of duplicate chains by filling the whole device until the total data reach to 50 GB with different wear leveling thresholds and collecting length of the longest chain

The results show that a duplicate chain is usually not too long. This is reasonable, since long duplicate chains indicate that there are too many invalid pages simultaneously present in the flash memory, which is usually avoided by a good design of the FTL and garbage collection

Evaluation and Simulation (cont.)

Throughput:

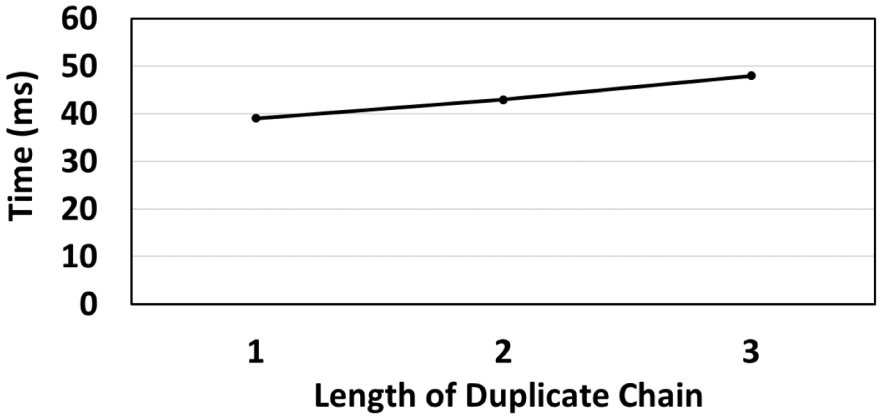


Conclusion:

- RedFlash has little influence on read operations. This is because, RedFlash does not need to modify any logic relating to reads
- RedFlash has influence on write operations. The write throughput of RedFlash decreases around 10% compared to that of OpenNFM. This is due to additional operations added to various functions of FTL, causing additional overhead to writes.

Evaluation and Simulation (cont.)

Overhead for secure deletion in SLC flash:



Conclusion:

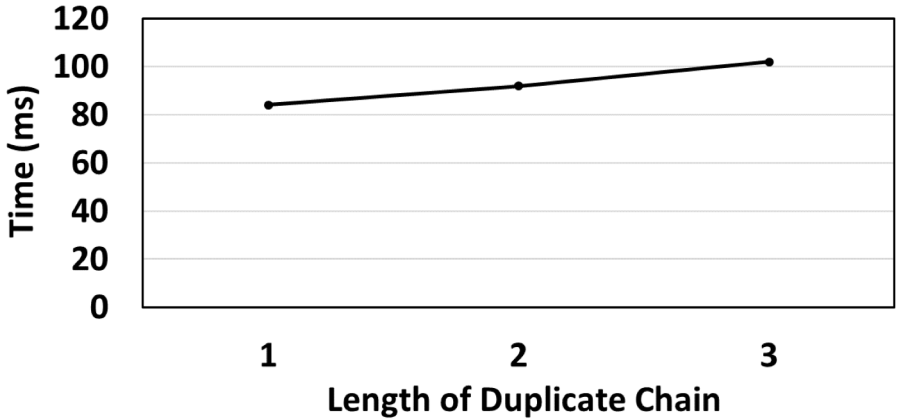
The time needed for secure deletion is linear to the length of the duplicate chain. This is reasonable, since RedFlash will remove both the targeted data and the corresponding duplicates upon secure deletion.

Evaluation and Simulation (cont.)

Simulation overhead for secure deletion in MLC flash:

Setup:

- Still rely on LPC-H3131
- Use a delay function to simulate read/program function



Conclusion:

- For MLC, the time needed for securely deleting a data node is linear with the length of its duplicate chain

Evaluation and Simulation (cont.)

Wear leveling:

Wear leveling threshold	WLI
10	1.87%
20	2.22%

WLI values under different wear leveling thresholds

$$WLI = \frac{1}{2}\sum_{i=1}^{n}\left\|\frac{e_i}{E} - \frac{1}{n}\right\|, E = \sum_{i=1}^{n} e_i$$

e_i: each block's erasure count

Wear Leveling Inequality (WLI) :

- Indicates fraction of erasures that must be reassigned to other blocks in order to achieve completely even wear
- Small WLI is an indication of good wear leveling

Thanks