Evanesco: Architectural Support for Efficient Data Sanitization in Modern Flash-Based Storage Systems

Myungsuk Kim*, Jisung Park*, Geonhee Cho, Yoona Kim, Lois Orosa, Onur Mutlu, and Jihong Kim

Seoul National University
SAFARI Research Group, ETH Zürich

ASPLOS 2020

*M. Kim and J. Park equally contributed.
Executive Summary

- **Motivation:** Secure deletion is essential in storage systems as modern computing systems process a large amount of security-sensitive data.

- **Problem:** It is challenging to support data sanitization in NAND flash-based SSDs.
  - Erase-before-write property → no overwrite on stored data
  - Physical data destruction → high performance & reliability overheads

- **Evanesco:** A low-cost data-sanitization technique w/o reliability issues
  - Uses on-chip access-control mechanisms instead of physically destroying data
  - Manages access-permission (AP) flags inside a NAND flash chip
    - Data is not accessible once the flash controller sets the data’s AP flag to disabled.
    - An AP flag cannot be reset before erasing the corresponding data.

- **Results**
  - Provides the same level of reliability as an unmodified SSD (w/o data-sanitization support)
    - Validated w/ 160 real state-of-the-art 3D NAND flash chips
  - Significantly improves performance and lifetime over existing data-sanitization techniques
    - Provides comparable (94.5%) performance with an unmodified SSD
Outline

- Secure Deletion in NAND Flash-Based SSDs

- Evanesco: Lock-Based Data Sanitization
  - pageLock: Page-Level Data Sanitization
  - blockLock: Block-Level Data Sanitization
  - SecureSSD: An Evanesco-Enabled SSD

- Evaluation

- Conclusion
Secure Deletion in Storage Systems

- Security-sensitive data is increasing in modern storage systems.

Confidential Data (e.g., Medical Record)
Secure Deletion in Storage Systems

- **Security-sensitive data is increasing** in modern storage systems.

---

Once a user deletes security-sensitive data, a storage system should guarantee its irrecoverability.
Data Versioning Problem

- Obsolete data in NAND flash-based solid-state drives (SSDs)
  - Old versions of updated or deleted files can remain in the SSD for a long time.

![Graph showing number of pages and time](graph.png)

- $f$: a heavily-updated DB file
- **# of valid pages of file $f$**
- **# of invalid pages of file $f$**

$t_0$: running out of free space
Data Versioning Problem

- Obsolete data in NAND flash-based solid-state drives (SSDs)
  - Old versions of updated or deleted files can remain in the SSD for a long time.

Updated or deleted data of a file can remain in SSDs due to unique features of NAND flash memory.
NAND Flash Memory Organization & Operations

Flash Cell

- Erased (1)
- Programmed (0)

NAND String

BitLine $BL_0$ $BL_1$ $BL_2$ $BL_{n-1}$
NAND Flash Memory Organization & Operations

**NAND String**

- Erased (1)
- Programmed (0)

**Page(s):** Unit of read and program (e.g., 131,072 cells: 16-KiB page)
**NAND Flash Memory Organization & Operations**

**NAND String**

- Flash Cell
  - Erased (1)
  - Programmed (0)

**Page(s):** Unit of read and program (e.g., 131,072 cells: 16-KiB page)
NAND Flash Memory Organization & Operations

**Page(s):** Unit of read and program (e.g., 131,072 cells: 16-KiB page)

- **Flash Cell:**
  - Erased (1)
  - Programmed (0)

- **NAND String:**

- **BitLine:** $BL_0$, $BL_1$, $BL_2$, $BL_{n-1}$
**NAND Flash Memory Organization & Operations**

**NAND String**

- **Flash Cell**
  - Erased (1)
  - Programmed (0)

**Page(s):** Unit of read and program (e.g., 131,072 cells: 16-KiB page)
NAND Flash Memory Organization & Operations

**Page(s):** Unit of read and program (e.g., 131,072 cells: 16-KiB page)
NAND Flash Memory Organization & Operations

Page(s): Unit of read and program (e.g., 131,072 cells: 16-KiB page)

Block: Erase unit (e.g., 576 pages: 9-MiB block)
NAND Flash Memory Organization & Operations

**Page(s):** Unit of read and program (e.g., 131,072 cells: 16-KiB page)

**Block:** Erase unit (e.g., 576 pages: 9-MiB block)

---

**NAND String**

- **Flash Cell**
  - Erased (1)
  - Programmed (0)

**BitLine**

- $B_{L_0}$
- $B_{L_1}$
- $B_{L_2}$
- $B_{L_{n-1}}$
NAND Flash Memory Organization & Operations

**Flash Cell**
- Erased (1)
- Programmed (0)

**NAND String**

**Page(s):** Unit of read and program (e.g., 131,072 cells: 16-KiB page)

**Block:** Erase unit (e.g., 576 pages: 9-MiB block)

**BitLine**
- BL₀
- BL₁
- BL₂
- BLₙ₋₁

**Peripherals**
- (Page Buffer, Decoder, ...)

Block#1

Block#2

...
**NAND Flash Memory Organization & Operations**

- **Page(s):** Unit of read and program (e.g., 131,072 cells: 16-KiB page)
- **Block:** Erase unit (e.g., 576 pages: 9-MiB block)

---

**Erase-before-write:** A block needs to be erased before programming a page (i.e., no overwrite on a page)
NAND Flash-Based SSD

Flash-Based SSD

File System

Flash Controller

Chip#0

0
1
2
3
Block#0

4
5
6
7
Page
Block#1

8
9
10
11
Block#2

12
13
14
15
Block#3

Chip#1

8
9
10
11
Block#2
12
13
14
15
Block#3
NAND Flash-Based SSD

Flash-Based SSD

Flash Translation Layer (FTL)

File System

Logical block-device view that supports overwrites
NAND Flash-Based SSD

Logical block-device view
that supports overwrites

Flash-Based SSD

Flash Translation Layer (FTL)

File System

A.png
B.db

0 1 2 3 4 5 6 7 8

A A A B B B

0 1 2 0 1 2

CTRL
DRAM
NAND NAND

Flash Controller

Chip#0

0 1 2 3

Block#0

4 5 6 7 Page

Chip#1

8 9 10 11

Block#2

12 13 14 15

Block#3

...
Flash-Based SSD

Flash Translation Layer (FTL)
- Address translation
  - Distributes host writes to fully exploit internal parallelism

Logical block-device view that supports overwrites
NAND Flash-Based SSD

Flash-Based SSD

Flash Translation Layer (FTL)

- Address translation
  - Distributes host writes to fully exploit internal parallelism

Logical block-device view that supports overwrites

Update B0 and B2

File System

Ctrl

DRAM

NAND NAND

Block#0

0 1 2 3 4 5 6 7

A0 A1 A2 B0' B1 B2'

Block#1

8 9 10 11

A1 B0 B2

Block#2

12 13 14 15

Block#3

Page
NAND Flash-Based SSD

Flash-Based SSD

Flash Translation Layer (FTL)

- Address translation
  - Distributes host writes to fully exploit internal parallelism
  - Out-of-place updates

Logical block-device view that supports overwrites

Update B0 and B2

File System

CTRL

DRAM

NAND

NAND

Logical block-device view that supports overwrites
NAND Flash-Based SSD

Flash-Based SSD

Flash Translation Layer (FTL)

- **Address translation**
  - Distributes host writes to fully exploit internal parallelism
  - Out-of-place updates

→ Logical-to-physical (L2P) mappings (e.g., LPA 1 → PPA 8)

Logical block-device view that supports overwrites

Logical Page Address

Physical Page Address

Update B0 and B2

File System

A.png

B.db

CTRL

DRAM

NAND

NAND

A0

A1

A2

B0

B1

B2

B0'

B1'

B2'

Flash Controller

Chip#0

0

A0

1

A2

2

B1

3

B0'

Block#0

4

5

6

7

Page

Block#1

Chip#1

8

A1

9

B0

10

B2

11

B2'

Block#2

12

13

14

15

Block#3
NAND Flash-Based SSD

**Flash-Based SSD**

**Flash Translation Layer (FTL)**

- **Address translation**
  - Distributes host writes to fully exploit internal parallelism
  - Out-of-place updates
  - Logical-to-physical (L2P) mappings (e.g., LPA 1 → PPA 8)

- **Garbage collection (GC)**
  - Reclams free pages for future host writes

**Logical block-device view that supports overwrites**
NAND Flash-Based SSD

Flash-Based SSD

Flash Translation Layer (FTL)
- **Address translation**
  - Distributes host writes to fully exploit internal parallelism
  - Out-of-place updates
  - Logical-to-physical (L2P) mappings (e.g., LPA 1 → PPA 8)
- **Garbage collection (GC)**
  - Reclaims free pages for future host writes
  - Selects a victim block with the smallest number of valid pages
  - Additional copy operations to move valid pages

Logical block-device view that supports overwrites
NAND Flash-Based SSD

Flash Translation Layer (FTL)

- **Address translation**
  - Distributes host writes to fully exploit internal parallelism
  - Out-of-place updates
  - Logical-to-physical (L2P) mappings (e.g., LPA 1 → PPA 8)

- **Garbage collection (GC)**
  - Reclaims free pages for future host writes
  - Selects a victim block with the smallest number of valid pages
  - Additional copy operations to move valid pages

Logical block-device view that supports overwrites
NAND Flash-Based SSD

Flash-Based SSD

Flash Translation Layer (FTL)
- **Address translation**
  - Distributes host writes to fully exploit internal parallelism
  - Out-of-place updates
  - **Logical-to-physical (L2P) mappings** (e.g., LPA 1 \(\rightarrow\) PPA 8)

- **Garbage collection (GC)**
  - Reclaims free pages for future host writes
  - Selects a victim block with the smallest number of valid pages
  - Additional copy operations to move valid pages

Logical block-device view that supports overwrites
NAND Flash-Based SSD

Flash-Based SSD

Flash Translation Layer (FTL)

- Address translation
  - Distributes host writes to fully exploit internal parallelism
  - Out-of-place updates
  - Logical-to-physical (L2P) mappings (e.g., LPA 1 → PPA 8)

- Garbage collection (GC)
  - Reclaims free pages for future host writes
  - Selects a victim block with the smallest number of valid pages
  - Additional copy operations to move valid pages
  - Page-status information (e.g., B0: invalid)

Logical block-device view that supports overwrites
# Data Deletion in NAND Flash-Based Storage Systems

## Flash-Based SSD

### Flash Translation Layer (FTL)

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>11</td>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PPA</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>valid</td>
</tr>
<tr>
<td>1</td>
<td>valid</td>
</tr>
<tr>
<td>2</td>
<td>valid</td>
</tr>
<tr>
<td>3</td>
<td>valid</td>
</tr>
<tr>
<td>4</td>
<td>valid</td>
</tr>
<tr>
<td>5</td>
<td>free</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>15</td>
<td>free</td>
</tr>
</tbody>
</table>

### L2P Mapping Table

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>11</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### Page Status Table

<table>
<thead>
<tr>
<th>PPA</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>valid</td>
</tr>
<tr>
<td>1</td>
<td>valid</td>
</tr>
<tr>
<td>2</td>
<td>valid</td>
</tr>
<tr>
<td>3</td>
<td>valid</td>
</tr>
<tr>
<td>4</td>
<td>valid</td>
</tr>
<tr>
<td>5</td>
<td>free</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>15</td>
<td>free</td>
</tr>
</tbody>
</table>

## File System

- A.png
- B.db

## Flash Controller

### Chip#0

<table>
<thead>
<tr>
<th>Block#0</th>
<th>Block#1</th>
<th>Block#2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>A1</td>
<td></td>
</tr>
<tr>
<td>A2</td>
<td></td>
<td>B2'</td>
</tr>
<tr>
<td>B1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Chip#1

<table>
<thead>
<tr>
<th>Block#0</th>
<th>Block#1</th>
<th>Block#2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

---

8
Data Deletion in NAND Flash-Based Storage Systems

Flash-Based SSD

Flash Translation Layer (FTL)

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>11</td>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PPA</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>valid</td>
</tr>
<tr>
<td>1</td>
<td>valid</td>
</tr>
<tr>
<td>2</td>
<td>valid</td>
</tr>
<tr>
<td>3</td>
<td>valid</td>
</tr>
<tr>
<td>4</td>
<td>valid</td>
</tr>
<tr>
<td>5</td>
<td>free</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>15</td>
<td>free</td>
</tr>
</tbody>
</table>

L2P Mapping Table

Page Status Table

File System

Delete A

Flash Controller

Chip#0

Block#0

Block#1

Chip#1

Block#2

Block#3
Data Deletion in NAND Flash-Based Storage Systems

Flash-Based SSD

Flash Translation Layer (FTL)

L2P Mapping Table

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>11</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Page Status Table

<table>
<thead>
<tr>
<th>PPA</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>valid</td>
</tr>
<tr>
<td>1</td>
<td>valid</td>
</tr>
<tr>
<td>2</td>
<td>valid</td>
</tr>
<tr>
<td>3</td>
<td>valid</td>
</tr>
<tr>
<td>4</td>
<td>valid</td>
</tr>
<tr>
<td>5</td>
<td>free</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>15</td>
<td>free</td>
</tr>
</tbody>
</table>

Delete A

File System

File: A.png
File: B.db

Flash Controller

Chip#0

<table>
<thead>
<tr>
<th>Block#0</th>
<th>Chip#1</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>A1</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>B2'</td>
</tr>
</tbody>
</table>

Flash Controller

Chip#0

<table>
<thead>
<tr>
<th>Block#0</th>
<th>Block#1</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>A1</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>B2'</td>
</tr>
</tbody>
</table>

Delete A
Data Deletion in NAND Flash-Based Storage Systems

Flash-Based SSD

Flash Translation Layer (FTL)

- **L2P Mapping Table**
  - LPA | PPA
  - 0   | N/A
  - 1   | N/A
  - 2   | N
  - 3   | 2
  - 4   | 12
  - ... | ...
  - 11  | N/A

- **Page Status Table**
  - PPA | Status
  - 0   | invalid
  - 1   | invalid
  - 2   | valid
  - 3   | valid
  - 4   | invalid
  - 5   | free
  - ... | ...
  - 15  | free

File System

- **Delete A**

- **Flash Controller**

- **Chip#0**
  - Block#0
    - 4: A1
  - Block#1
    - 3: B1
  - Block#2
    - 12: B2

- **Chip#1**
  - Block#2
    - 12: B2
Data Deletion in NAND Flash-Based Storage Systems

Flash-Based SSD

Flash Translation Layer (FTL)

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
<td>invalid</td>
</tr>
<tr>
<td>1</td>
<td>N/A</td>
<td>invalid</td>
</tr>
<tr>
<td>2</td>
<td>N/A</td>
<td>valid</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>valid</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>invalid</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>free</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>N/A</td>
<td>invalid</td>
</tr>
</tbody>
</table>

L2P Mapping Table

Page Status Table

File System

Update B1

Flash Controller

Chip#0

Block#0

Block#1

Chip#1

Block#2

Block#3
Data Deletion in NAND Flash-Based Storage Systems

Flash-Based SSD

Flash Translation Layer (FTL)

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>1</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>N/A</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>11</td>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PPA</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>invalid</td>
</tr>
<tr>
<td>1</td>
<td>invalid</td>
</tr>
<tr>
<td>2</td>
<td>valid</td>
</tr>
<tr>
<td>3</td>
<td>valid</td>
</tr>
<tr>
<td>4</td>
<td>invalid</td>
</tr>
<tr>
<td>5</td>
<td>free</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>15</td>
<td>free</td>
</tr>
</tbody>
</table>

L2P Mapping Table

Page Status Table

File System

Update B1

Out-of-place update
Data Deletion in NAND Flash-Based Storage Systems

Flash-Based SSD

Flash Translation Layer (FTL)

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>1</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>N/A</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>11</td>
<td>N/A</td>
</tr>
</tbody>
</table>

L2P Mapping Table

<table>
<thead>
<tr>
<th>PPA</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>invalid</td>
</tr>
<tr>
<td>1</td>
<td>invalid</td>
</tr>
<tr>
<td>2</td>
<td>invalid</td>
</tr>
<tr>
<td>3</td>
<td>valid</td>
</tr>
<tr>
<td>4</td>
<td>invalid</td>
</tr>
<tr>
<td>5</td>
<td>free</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>15</td>
<td>free</td>
</tr>
</tbody>
</table>

Page Status Table

File System

Update B1

Out-of-place update

LPA: Logical Page Address
PPA: Physical Page Address

Update status

Flash Controller

Chip#0

<table>
<thead>
<tr>
<th>Block#0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 A0</td>
</tr>
<tr>
<td>1 A2</td>
</tr>
<tr>
<td>2 B1</td>
</tr>
<tr>
<td>3 B0'</td>
</tr>
</tbody>
</table>

Chip#1

<table>
<thead>
<tr>
<th>Block#2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 A1</td>
</tr>
<tr>
<td>5 B1'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block#3</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
</tr>
<tr>
<td>12 B2'</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>15</td>
</tr>
</tbody>
</table>
Flash-Based SSD

Flash Translation Layer (FTL)

L2P Mapping Table

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>1</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>N/A</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Page Status Table

<table>
<thead>
<tr>
<th>PPA</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>invalid</td>
</tr>
<tr>
<td>1</td>
<td>invalid</td>
</tr>
<tr>
<td>2</td>
<td>invalid</td>
</tr>
<tr>
<td>3</td>
<td>valid</td>
</tr>
<tr>
<td>4</td>
<td>invalid</td>
</tr>
<tr>
<td>5</td>
<td>free</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>15</td>
<td>free</td>
</tr>
</tbody>
</table>

File System

File system structure:
- A.png
- B.db

Update B1

Update status

Update mapping

Out-of-place update

Flash Controller

Chip#0
- 0 A0
- 1 A2
- 2 B1
- 3 B0'

Block#0
- 4 A1
- 5 B1'

Chip#1
- 8 B2'

Block#2
- 10 B0'
- 11 B1'

Block#3
- 12 B2'
- 13 B1'
- 15 B0'
Data Deletion in NAND Flash-Based Storage Systems

Invalid data remains in NAND flash chips until GC erases the corresponding block(s)
Security Vulnerability of NAND Flash-Based SSDs

Flash-Based SSD

Flash Translation Layer (FTL)

L2P Mapping Table

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>1</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>N/A</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>11</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Page Status Table

<table>
<thead>
<tr>
<th>PPA</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>invalid</td>
</tr>
<tr>
<td>1</td>
<td>invalid</td>
</tr>
<tr>
<td>2</td>
<td>invalid</td>
</tr>
<tr>
<td>3</td>
<td>valid</td>
</tr>
<tr>
<td>4</td>
<td>invalid</td>
</tr>
<tr>
<td>5</td>
<td>free</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>15</td>
<td>free</td>
</tr>
</tbody>
</table>

File System

Block#0

<table>
<thead>
<tr>
<th>Block#0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
</tr>
<tr>
<td>A2</td>
</tr>
<tr>
<td>B1</td>
</tr>
<tr>
<td>B0’</td>
</tr>
</tbody>
</table>

Block#1

<table>
<thead>
<tr>
<th>Block#1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
</tr>
<tr>
<td>B1’</td>
</tr>
</tbody>
</table>

Block#2

<table>
<thead>
<tr>
<th>Block#2</th>
</tr>
</thead>
<tbody>
<tr>
<td>B2’</td>
</tr>
</tbody>
</table>

Block#3

<table>
<thead>
<tr>
<th>Block#3</th>
</tr>
</thead>
</table>

Flash Controller

Chip#0

<table>
<thead>
<tr>
<th>Chip#0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

Chip#1

<table>
<thead>
<tr>
<th>Chip#1</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
</tbody>
</table>

Adversary
Security Vulnerability of NAND Flash-Based SSDs
Security Vulnerability of NAND Flash-Based SSDs

Flash-Based SSD

Flash Translation Layer (FTL)

L2P Mapping Table

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>1</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>N/A</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>11</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Page Status Table

<table>
<thead>
<tr>
<th>PPA</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>invalid</td>
</tr>
<tr>
<td>1</td>
<td>invalid</td>
</tr>
<tr>
<td>2</td>
<td>invalid</td>
</tr>
<tr>
<td>3</td>
<td>valid</td>
</tr>
<tr>
<td>4</td>
<td>invalid</td>
</tr>
<tr>
<td>5</td>
<td>free</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>15</td>
<td>free</td>
</tr>
</tbody>
</table>

Flash Controller

Block#0

<table>
<thead>
<tr>
<th>Chip#0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

Block#1

<table>
<thead>
<tr>
<th>Chip#1</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
</tbody>
</table>

Block#2

<table>
<thead>
<tr>
<th>Block#2</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
</tbody>
</table>
Security Vulnerability of NAND Flash-Based SSDs

Flash-Based SSD

Flash Translation Layer (FTL)

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>1</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>N/A</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>11</td>
<td>N/A</td>
</tr>
</tbody>
</table>

No mappings to invalid PPAs

<table>
<thead>
<tr>
<th>PPA</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>invalid</td>
</tr>
<tr>
<td>1</td>
<td>invalid</td>
</tr>
<tr>
<td>2</td>
<td>invalid</td>
</tr>
<tr>
<td>3</td>
<td>valid</td>
</tr>
<tr>
<td>4</td>
<td>invalid</td>
</tr>
<tr>
<td>5</td>
<td>free</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>15</td>
<td>free</td>
</tr>
</tbody>
</table>

L2P Mapping Table

Page Status Table

Direct access to SSD
Security Vulnerability of NAND Flash-Based SSDs
Security Vulnerability of NAND Flash-Based SSDs

![Diagram of Custom Flash Controller and NAND Chips]

- **Custom Flash Controller**
- **Direct access to raw NAND chips**
- **De-solder**

Diagram showing connections and layout of NAND chips and blocks.
Security Vulnerability of NAND Flash-Based SSDs

ADVERSARY

Entire Deleted File

Previous Ver. of Updated File

Forensic Tool

Custom Flash Controller

Direct access to raw NAND chips

De-solder

NAND NAND

Chip#0

A0 A2 A1 B1

Block#0

4 5 6 7

Chip#1

8 9 10 11

Block#2

12 13 14 15

Block#3

Entire Deleted File

Previous Ver. of Updated File

Forensic Tool

Custom Flash Controller

Direct access to raw NAND chips

De-solder

NAND NAND

Chip#0

A0 A2 A1 B1

Block#0

4 5 6 7

Chip#1

8 9 10 11

Block#2

12 13 14 15

Block#3
Security Vulnerability of NAND Flash-Based SSDs

Deleted or updated files can be recovered by *directly accessing* raw NAND flash chips.
Existing Solution: Immediate Block Erasure

- Immediately erases the block that stores data to be sanitized
Existing Solution: Immediate Block Erasure

- Immediately erases the block that stores data to be sanitized
  - High performance and lifetime overheads due to *Erase-before-write property*
  - Needs to copy all the valid pages stored in the same block
Existing Solution: Immediate Block Erasure

- Immediately erases the block that stores data to be sanitized
  - High performance and lifetime overheads due to Erase-before-write property
    - Needs to copy all the valid pages stored in the same block

576 Pages

Target Block

Free
Valid
Invalid

Free Block
### Existing Solution: Immediate Block Erasure

- Immediately erases the block that stores data to be sanitized
  - High performance and lifetime overheads due to *Erase-before-write property*
  - Needs to copy all the valid pages stored in the same block

![Diagram showing target block and free block with formulas for time calculation: \( t = N_{\text{copy}} \times (t_{\text{READ}} + t_{\text{PROG}}) \).]
Existing Solution: Immediate Block Erasure

- Immediately erases the block that stores data to be sanitized
  - High performance and lifetime overheads due to Erase-before-write property
    - Needs to copy all the valid pages stored in the same block
Existing Solution: Immediate Block Erasure

- Immediately erases the block that stores data to be sanitized
  - High performance and lifetime overheads due to *Erase-before-write property*
    - Needs to copy all the valid pages stored in the same block

\[ t = N_{copy} \times (t_{READ} + t_{PROG}) \]
Existing Solution: Immediate Block Erasure

- Immediately erases the block that stores data to be sanitized
  - High performance and lifetime overheads due to Erase-before-write property
    - Needs to copy all the valid pages stored in the same block

Immediate block erasure:
High performance and lifetime overheads
Existing Solution: Reprogramming the Page

- Scrubbing [Wei+, FAST’2011]: Reprograms all the flash cells storing an invalid page
  - Destroys the page data w/o block erasure
Existing Solution: Reprogramming the Page

- Scrubbing [Wei+, FAST’2011]: Reprograms all the flash cells storing an invalid page
  - Destroys the page data w/o block erasure
Existing Solution: Reprogramming the Page

- Scrubbing [Wei+, FAST’2011]: Reprograms all the flash cells storing an invalid page
  - Destroys the page data w/o block erasure
  - Performance and lifetime overheads in *Multi-level cell* (MLC) NAND flash memory
    - Needs to copy all the valid pages stored in the same flash cells
Existing Solution: Reprogramming the Page

- Scrubbing [Wei+, FAST’2011]: Reprograms all the flash cells storing an invalid page
  - Destroys the page data w/o block erasure
  - Performance and lifetime overheads in *Multi-level cell* (MLC) NAND flash memory
    - Needs to copy all the valid pages stored in the same flash cells
Existing Solution: Reprogramming the Page

- **Scrubbing [Wei+, FAST’2011]:** Reprograms all the flash cells storing an invalid page
  - Destroys the page data \textit{w/o} block erasure
  - Performance and lifetime overheads in \textit{Multi-level cell} (MLC) NAND flash memory
    - Needs to \textit{copy} all the valid pages stored in the same flash cells
  - Reliability issues: \textit{cell-to-cell interference}

![Diagram showing share flash cells, target block, target page, cell-to-cell interference]
Existing Solution: Reprogramming the Page

- Scrubbing [Wei+, FAST’2011]: Reprograms all the flash cells storing an invalid page
  - Destroys the page data w/o block erasure
  - Performance and lifetime overheads in Multi-level cell (MLC) NAND flash memory
    - Needs to copy all the valid pages stored in the same flash cells
  - Reliability issues: cell-to-cell interference

Existing solutions incur performance, lifetime, and reliability problems in modern NAND flash memory
Outline

- Secure Deletion in NAND Flash-Based SSDs

- Evanesco: Lock-Based Data Sanitization
  - pageLock: Page-Level Data Sanitization
  - blockLock: Block-Level Data Sanitization
  - SecureSSD: An Evanesco-Enabled SSD

- Evaluation

- Conclusion
**Key idea:** Allow a NAND flash chip to be aware of data validity

- Prevent access to invalid data at the chip level w/o destroying the data
  - Low overhead: No copy operation to move valid pages stored in the same cells
  - High reliability: No cell-to-cell interference to other valid pages

---

**Evanesco:** Access Control-Based Sanitization

Diagram showing NAND闪存的区块和标志位：

- **NAND**
  - Block#0
    - A0
    - A1
    - A2
    - B0
  - Flags#0: E (Enabled), D (Disabled)
  - Block#1
  - Flags#1: E (Enabled)
**Evanesco: Access Control-Based Sanitization**

- **Key idea:** Allow a NAND flash chip to be aware of data validity
  - Prevent access to invalid data **at the chip level** w/o destroying the data
    → Low overhead: **No copy operation** to move valid pages stored in the same cells
    → High reliability: **No cell-to-cell interference** to other valid pages

- Two **new NAND flash commands**: pageLock (pLock) and blockLock (bLock)
  - **pLock:** disables access to a page
  - **bLock:** disables access to all the page in a block
**Evanesco: Access Control-Based Sanitization**

- **Key idea:** Allow a NAND flash chip to be aware of data validity
  - Prevent access to invalid data at the chip level w/o destroying the data
    - Low overhead: No copy operation to move valid pages stored in the same cells
    - High reliability: No cell-to-cell interference to other valid pages

- **Two new NAND flash commands:** pageLock (pLock) and blockLock (bLock)
  - pLock: disables access to a page
  - bLock: disables access to all the page in a block
Key idea: Allow a NAND flash chip to be aware of data validity
- Prevent access to invalid data at the chip level w/o destroying the data
  - Low overhead: No copy operation to move valid pages stored in the same cells
  - High reliability: No cell-to-cell interference to other valid pages

Two new NAND flash commands: pageLock (pLock) and blockLock (bLock)
- pLock: disables access to a page
- bLock: disables access to all the page in a block

Evanesco: Access Control-Based Sanitization
**Key idea:** Allow a NAND flash chip to be aware of data validity
- Prevent access to invalid data at the chip level w/o destroying the data
  - Low overhead: No copy operation to move valid pages stored in the same cells
  - High reliability: No cell-to-cell interference to other valid pages

**Two new NAND flash commands:** pageLock (pLock) and blockLock (bLock)
- **pLock:** disables access to a page
- **bLock:** disables access to all the page in a block

---

**Evanesco: Access Control-Based Sanitization**

- **E:** Enabled, **D:** Disabled

---

<table>
<thead>
<tr>
<th>NAND</th>
<th>Flags#0</th>
<th>Flags#1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>1</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>2</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>4</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>5</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>7</td>
<td>E</td>
<td>E</td>
</tr>
</tbody>
</table>

**pLock (3)**
**Evanesco: Access Control-Based Sanitization**

- **Key idea:** Allow a NAND flash chip to be aware of data validity
  - Prevent access to invalid data at the chip level w/o destroying the data
    - Low overhead: No copy operation to move valid pages stored in the same cells
    - High reliability: No cell-to-cell interference to other valid pages

- **Two new NAND flash commands:** pageLock (pLock) and blockLock (bLock)
  - **pLock:** disables access to a page
  - **bLock:** disables access to all the page in a block
Key idea: Allow a NAND flash chip to be aware of data validity
- Prevent access to invalid data at the chip level w/o destroying the data
  → Low overhead: No copy operation to move valid pages stored in the same cells
  → High reliability: No cell-to-cell interference to other valid pages

Two new NAND flash commands: pageLock (pLock) and blockLock (bLock)
- pLock: disables access to a page
- bLock: disables access to all the page in a block
**Key idea:** Allow a NAND flash chip to be aware of data validity
- Prevent access to invalid data at the chip level w/o destroying the data
  → Low overhead: No copy operation to move valid pages stored in the same cells
  → High reliability: No cell-to-cell interference to other valid pages

**Two new NAND flash commands:** pageLock (pLock) and blockLock (bLock)
- **pLock:** disables access to a page
- **bLock:** disables access to all the page in a block

---

![Diagram of NAND flash setup](image)
Outline

- Secure Deletion in NAND Flash-Based SSDs
- Evanesco: Lock-Based Data Sanitization
  - pageLock: Page-Level Data Sanitization
  - blockLock: Block-Level Data Sanitization
  - SecureSSD: An Evanesco-Enabled SSD
- Evaluation
- Conclusion
pLock: Page-Level Data Sanitization

- Implements page access-permission (pAP) flags using spare cells
  - Sets a pAP flag to disabled (enabled) by programming (erasing) the flag cells
    → A disabled page cannot be enabled until the entire block is erased.
  - No additional command to access a pAP flag: read with the page data at the same time
pLock: Page-Level Data Sanitization

- Implements page access-permission (pAP) flags using spare cells
  - Sets a pAP flag to disabled (enabled) by programming (erasing) the flag cells
    → A disabled page cannot be enabled until the entire block is erased.
  - No additional command to access a pAP flag: read with the page data at the same time
pLock: Page-Level Data Sanitization

- Implements **page access-permission (pAP) flags using spare cells**
  - Sets a pAP flag to disabled (enabled) by programming (erasing) the flag cells
    - A disabled page cannot be enabled until the entire block is erased.
  - No additional command to access a pAP flag: read with the page data at the same time
pLock: Page-Level Data Sanitization

- Implements page access-permission (pAP) flags using spare cells
  - Sets a pAP flag to disabled (enabled) by programming (erasing) the flag cells
    → A disabled page cannot be enabled until the entire block is erased.
  - No additional command to access a pAP flag: read with the page data at the same time
- Prevents transfer of data from a disabled page
  - The bridge transistor disconnects the page buffer from the data-out circuitry.
pLock: Page-Level Data Sanitization

- Implements page access-permission (pAP) flags using spare cells
  - Sets a pAP flag to disabled (enabled) by programming (erasing) the flag cells
    - A disabled page cannot be enabled until the entire block is erased.
  - No additional command to access a pAP flag: read with the page data at the same time
- Prevents transfer of data from a disabled page
  - The bridge transistor disconnects the page buffer from the data-out circuitry.

### Diagram

- **Data Area**
- **Spare Area (for metadata)**
- **Flash Cell**: Erased (1) Programmed (0)
- **Flag Cell**: 1: Enabled 0: Disabled
- **Page Buffer**
- **Bitline (BL)**: Read (2)
- **Block**
- **Page Buffer**
- **Data Out**
- **Bridge Transistor**
pLock: Page-Level Data Sanitization

- Implements page access-permission (pAP) flags using spare cells
  - Sets a pAP flag to disabled (enabled) by programming (erasing) the flag cells
    → A disabled page cannot be enabled until the entire block is erased.
  - No additional command to access a pAP flag: read with the page data at the same time
- Prevents transfer of data from a disabled page
  - The bridge transistor disconnects the page buffer from the data-out circuitry.
pLock: Page-Level Data Sanitization

- Implements **page access-permission (pAP) flags using spare cells**
  - Sets a pAP flag to disabled (enabled) by programming (erasing) the flag cells
    - A disabled page cannot be enabled until the entire block is erased.
  - No additional command to access a pAP flag: read with the page data at the same time
- Prevents transfer of data from a disabled page
  - The bridge transistor disconnects the page buffer from the data-out circuitry.
pLock: Implementation Details

Data Cells

pAP Flags

Majority Circuit

AP Flag Selector

Page Buffer

Bridge Transistor

Data Out
Problem 1: Multiple pages are stored in the same flash cell.
- Solution: Use multiple flags for each row (e.g., 3 flags for triple-level cell (TLC) NAND)
pLock: Implementation Details

- **Problem 1:** Multiple pages are stored in the same flash cell.
  - **Solution:** Use multiple flags for each row (e.g., 3 flags for triple-level cell (TLC) NAND)
- **Problem 2:** A flag cell can misbehave → unintentional disabling or enabling of a page
  - **Solution:** Use multiple flag cells for each pAP flag (k-modular redundancy scheme)

![Diagram of pLock implementation details]

- **Data Cells:**
  - MSB
  - CSB
  - LSB
- **pAP Flags**
- **AP Flag Selector**
- **Majority Circuit**
- **Page Buffer**
- **Bridge Transistor**
- **Data Out**

**Diagram Notes:**
- BL<sub>n-1</sub>, BL<sub>n</sub>, BL<sub>n+k</sub>, BL<sub>n+2k</sub>, BL<sub>n+3k-1</sub>
- k modular redundancy scheme
- LSB, CSB, MSB
**Problem 1:** Multiple pages are stored in the same flash cell.
- **Solution:** Use multiple flags for each row (e.g., 3 flags for triple-level cell (TLC) NAND)

**Problem 2:** A flag cell can misbehave → unintentional disabling or enabling of a page
- **Solution:** Use multiple flag cells for each pAP flag ($k$-modular redundancy scheme)

---

**pLock: Implementation Details**

![Diagram of pLock implementation](attachment:diagram.png)

- **Data Out Bridge Transistor**
- **Data Cells**
- **pAP Flags**
- **AP Flag Selector**
- **Majority Circuit**
- **Page Buffer**
- **Block#k**
- **No cell-to-cell interference**
Problem 1: Multiple pages are stored in the same flash cell.
- **Solution:** Use multiple flags for each row (e.g., 3 flags for triple-level cell (TLC) NAND)

Problem 2: A flag cell can misbehave → unintentional disabling or enabling of a page
- **Solution:** Use multiple flag cells for each pAP flag (k-modular redundancy scheme)

Reliability issues
1. Cell-to-cell interference b/w flag cells in the same NAND strings
2. Program disturbance due to high program voltage to the other cells at the same row
Problem 1: Multiple pages are stored in the same flash cell.
- Solution: Use multiple flags for each row (e.g., 3 flags for triple-level cell (TLC) NAND)

Problem 2: A flag cell can misbehave → unintentional disabling or enabling of a page
- Solution: Use multiple flag cells for each pAP flag ($k$-modular redundancy scheme)

Reliability issues
1. Cell-to-cell interference b/w flag cells in the same NAND strings
2. Program disturbance due to high program voltage to the other cells at the same row

Solutions
1. Use flag cells in single-level cell (SLC) mode
   - More robust to interference and disturbance
   - Reduces pLock latency
2. One-shot programming w/ low voltage
   - Reduces interference and disturbance
Problem 1: Multiple pages are stored in the same flash cell.
- Solution: Use multiple flags for each row (e.g., 3 flags for triple-level cell (TLC) NAND)

Problem 2: A flag cell can misbehave → unintentional disabling or enabling of a page
- Solution: Use multiple flag cells for each pAP flag (k-modular redundancy scheme)

Reliability issues
1. Cell-to-cell interference b/w flag cells in the same NAND strings
2. Program disturbance due to high program voltage to the other cells at the same row

Solutions
1. Use flag cells in single-level cell (SLC) mode
   - More robust to interference and disturbance

pLock: Prevents data transfer for a disabled page
→ Reliable and copy-free per-page sanitization
Outline

- Secure Deletion in NAND Flash-Based SSDs

- **Evanesco: Lock-Based Data Sanitization**
  - `pageLock`: Page-Level Data Sanitization
  - `blockLock`: Block-Level Data Sanitization
  - `SecureSSD`: An Evanesco-Enabled SSD

- Evaluation

- Conclusion
Problem with Page-Level Sanitization

- Nontrivial performance overhead in invalidating an entire block
  - Deleting a 1-GiB video → 65,536 pLock operations (page size = 16 KiB)
  - Invalidating blocks in SSD management tasks (GC, wear-leveling, ...)

Problem with Page-Level Sanitization

- Nontrivial performance overhead in invalidating an entire block
  - Deleting a 1-GiB video $\rightarrow$ 65,536 pLock operations (page size = 16 KiB)
  - Invalidating blocks in SSD management tasks (GC, wear-leveling, ...)

- Immediate block erasure is not feasible in 3D NAND flash memory.
  - Open-block problem: Reliability degradation due to a long time interval b/w erasing and programming a block $\rightarrow$ A block should be erased lazily.
Problem with Page-Level Sanitization

- Nontrivial performance overhead in invalidating an entire block
  - Deleting a 1-GiB video \(\rightarrow\) 65,536 pLock operations (page size = 16 KiB)
  - Invalidating blocks in SSD management tasks (GC, wear-leveling, ...)

- Immediate block erasure is not feasible in 3D NAND flash memory.
  - Open-block problem: Reliability degradation due to a long time interval b/w erasing and programming a block \(\rightarrow\) A block should be erased lazily.

![Normalized RBER vs. Length of open interval graph]

- No P/E cycling
- After P/E cycling
- After P/E cycling + retention

<table>
<thead>
<tr>
<th>Length of open interval</th>
<th>Zero</th>
<th>Very short</th>
<th>Short</th>
<th>Medium</th>
<th>Long</th>
<th>Very long</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normalized RBER (Raw Bit Error Rate)</td>
<td>0.8</td>
<td>1.0</td>
<td>1.2</td>
<td>1.4</td>
<td>1.2</td>
<td>1.4</td>
</tr>
</tbody>
</table>
Problem with Page-Level Sanitization

- Nontrivial performance overhead in invalidating an entire block
  - Deleting a 1-GiB video → 65,536 pLock operations (page size = 16 KiB)
  - Invalidating blocks in SSD management tasks (GC, wear-leveling, ...)

- Immediate block erasure is not feasible in 3D NAND flash memory.
  - Open-block problem: Reliability degradation due to a long time interval b/w erasing and programming a block → A block should be erased lazily.

![Diagram showing the impact of P/E cycling and retention on RBER](chart.png)
**bLock: Block-Level Sanitization**

- **Key idea:** Program the *string-select line (SSL)* of a block
**Key idea:** Program the *string-select line (SSL)* of a block

- 3D NAND flash memory implements an SSL using flash cells.
**bLock: Block-Level Sanitization**

- **Key idea:** Program the *string-select line (SSL)* of a block
  - 3D NAND flash memory implements an SSL using flash cells.
**Key idea:** Program the *string-select line (SSL)* of a block

- 3D NAND flash memory implements an SSL using flash cells.
Key idea: Program the string-select line (SSL) of a block

- 3D NAND flash memory implements an SSL using flash cells.
- SSL programming: Disconnects all the pages from bitlines (i.e., from the page buffer)
**Key idea:** Program the *string-select line (SSL)* of a block

- 3D NAND flash memory implements an SSL using flash cells.
- SSL programming: Disconnects all the pages from bitlines (i.e., from the page buffer)

---

**bLock: Block-Level Sanitization**

![Diagram](image-url)
**Key idea:** Program the *string-select line (SSL)* of a block

- 3D NAND flash memory implements an SSL using flash cells.
- SSL programming: Disconnects all the pages from bitlines (i.e., from the page buffer)

![Diagram of Block-Level Sanitization](image-url)
bLock: Block-Level Sanitization

- **Key idea:** Program the *string-select line (SSL)* of a block
  - 3D NAND flash memory implements an SSL using flash cells.
  - SSL programming: Disconnects all the pages from bitlines (i.e., from the page buffer)
bLock: Block-Level Sanitization

- **Key idea:** Program the *string-select line (SSL)* of a block
  - 3D NAND flash memory implements an SSL using flash cells.
  - SSL programming: Disconnects all the pages from bitlines (i.e., from the page buffer)

![Diagram showing Block-Level Sanitization](image)
**bLock: Block-Level Sanitization**

- **Key idea:** Program the *string-select line (SSL)* of a block
  - 3D NAND flash memory implements an SSL using flash cells.
  - SSL programming: Disconnects all the pages from bitlines (i.e., from the page buffer)

![Diagram](image)

- **Block#0**
  - No voltage (deactivate)

- **Block#k**
  - $V_{ref}$ (activate)
  - Cannot activate w/ normal voltage
  - Disconnected from BLs
  - $\rightarrow$ No current through strings

- **Block#N-1**
  - No voltage (deactivate)
  - All-zero data

- **Page Buffer**
  - Data Out
Key idea: Program the string-select line (SSL) of a block

- 3D NAND flash memory implements an SSL using flash cells.
- SSL programming: Disconnects all the pages from bitlines (i.e., from the page buffer)

bLock: Block-Level Sanitization

bLock: Programs the SSL of block

→ Disconnects all the pages from bitlines until the block is physically erased
Outline

- Secure Deletion in NAND Flash-Based SSDs

- **Evanesco: Lock-Based Data Sanitization**
  - pageLock: Page-Level Data Sanitization
  - blockLock: Block-Level Data Sanitization
  - SecureSSD: An Evanesco-Enabled SSD

- Evaluation

- Conclusion
SecureSSD: An Evanesco-Enabled SSD

- An SSD that supports immediate data sanitization of updated or deleted data
  - Lock manager issues pLock and bLock commands depending on the block’s status.
SecureSSD: An Evanesco-Enabled SSD

- An SSD that supports **immediate data sanitization** of updated or deleted data
  - Lock manager issues pLock and bLock commands depending on the block’s status.

```
Application

Delete A

File System

A.png  B.doc  ...

Trim (discard) LPAs

SecureSSD

Evanesco-Aware FTL

L2P Mapping Table  Page Status Table  Lock Manager

Flash Chip

A.png  B.doc
```
SecureSSD: An Evanesco-Enabled SSD

- An SSD that supports immediate data sanitization of updated or deleted data
  - Lock manager issues pLock and bLock commands depending on the block’s status.

```
Application

Delete A

File System

Trim (discard) LPAs

SecureSSD

Evanesco-Aware FTL

L2P Mapping Table
Page Status Table
Lock Manager

Flash Chip

pLock
bLock
```
SecureSSD: An Evanesco-Enabled SSD

- An SSD that supports **immediate data sanitization** of updated or deleted data
  - Lock manager issues pLock and bLock commands depending on the block’s status.

![Diagram of SecureSSD architecture](image)

- **Application**
- **File System**
- **SecureSSD**
  - **Evanesco-Aware FTL**
    - L2P Mapping Table
    - Page Status Table
  - **Lock Manager**
  - **Flash Chip**

### Diagram Details
- **Delete A**
- **Trim (discard) LPAs**
- **Block#k**
  - Valid
  - Invalid

- **pLock**
- **bLock**
SecureSSD: An Evanesco-Enabled SSD

- An SSD that supports **immediate data sanitization** of updated or deleted data
  - Lock manager issues pLock and bLock commands depending on the block’s status.

![Diagram of SecureSSD](image)

**Evanesco-Aware FTL**
- L2P Mapping Table
- Page Status Table
- Lock Manager

**Flash Chip**
- A.png
- B.doc

**Valid**
**Invalid**
SecureSSD: An Evanesco-Enabled SSD

- An SSD that supports **immediate data sanitization** of updated or deleted data
  - Lock manager issues pLock and bLock commands depending on the block’s status.

---

**Evanesco-Aware FTL**

- L2P Mapping Table
- Page Status Table
- Lock Manager

**Flash Chip**

- A.png
- B.doc

**Application**

- Delete A

**File System**

- A.png
- B.doc
- ...

**SecureSSD**

- Multi invalid pages + no valid pages in the block

**Block#k**

- Valid
- Invalid

**Block#m**

- Valid
- Invalid

**Delete**

- Trim (discard) LPAs
SecureSSD: An Evanesco-Enabled SSD

- An SSD that supports **immediate data sanitization** of updated or deleted data
  - Lock manager issues `pLock` and `bLock` commands depending on the block’s status.

---

**Application**

- **Delete A**

**File System**

- **Trim (discard) LPAs**

**SecureSSD**

- **Evanesco-Aware FTL**
  - L2P Mapping Table
  - Page Status Table
  - Lock Manager

- **Flash Chip**
  - `pLock` and `bLock` commands

**Block**

- **Block#k**
  - Multiple invalid pages + no valid pages in the block

- **Block#m**
SecureSSD: Selective Data Sanitization

- SecureSSD avoids unnecessary pLock and bLock for security-insensitive data.
  - A user sets the security requirements of written data w/ extended I/O interfaces.
SecureSSD: Selective Data Sanitization

- SecureSSD avoids unnecessary pLock and bLock for security-insensitive data.
  - A user sets the security requirements of written data with extended I/O interfaces.

```c
fd = open("B.doc", O_CREATE|O_INSEC);
// create security-insensitive file B.doc
bio->bi_of |= REQ_OP_INSEC_WRITE;
// set low security requirement
```

Set page status to INSECURE
SecureSSD: Selective Data Sanitization

- SecureSSD avoids unnecessary pLock and bLock for security-insensitive data.
  - A user sets the security requirements of written data with extended I/O interfaces.

![Diagram](image.png)
SecureSSD: Selective Data Sanitization

- SecureSSD avoids unnecessary pLock and bLock for security-insensitive data.
  - A user sets the security requirements of written data with extended I/O interfaces.

![Diagram of SecureSSD system]

- **Application**
  - *Delete B*

- **File System**
  - *Trim (discard) LPAs*

- **SecureSSD**
  - **Evanesco-Aware FTL**
    - L2P Mapping Table
    - Extended PG Status Table
    - Lock Manager
  - **Flash Chip**
    - Invalidation w/o pLock or bLock

- **Set page status to INSECURE**
SecureSSD: Selective Data Sanitization

- SecureSSD avoids unnecessary pLock and bLock for security-insensitive data.
  - A user sets the security requirements of written data with extended I/O interfaces.

**Diagram:**
- **Application**
  - Delete B
- **File System**
  - Trim (discard) LPAs
- **SecureSSD**
  - Evanesco-Aware FTL
  - Flash Chip

**SecureSSD minimizes data-sanitization overheads**
Outline

- Secure Deletion in NAND Flash-Based SSDs
- Evanesco: Lock-Based Data Sanitization
  - pageLock: Page-Level Data Sanitization
  - blockLock: Block-Level Data Sanitization
  - SecureSSD: An Evanesco-Enabled SSD

- Evaluation

- Conclusion
Methodology

- **Design space exploration for pLock and bLock**
  - Using 160 real state-of-the-art 3D triple-level-cell (TLC) NAND flash chips
  - To find the best operation parameters w/o reliability degradation
    - **pLock**: 100-us latency w/ 9 flag cells per page
    - **bLock**: 300-us latency
    - tREAD = 100 us, tPROG = 700 us, tBERS = 3.5 ms

- **Simulator**: Open SSD-development platform (FlashBench [Lee+, RSP’2012])
  - 32-GiB storage capacity
  - 576 pages per block
  - 16-KiB page size

- **Compared SSDs**
  - **erSSD**: Erases the entire block after copying valid pages in the block
  - **scrSSD**: Performs scrubbing after copying valid pages in the same cells [Wei+, FAST’2011]

- **Workloads**
  - Three server workloads: MailServer, DBServer, FileServer
  - Mobile workload collected from an Android smartphone (Samsung Galaxy S2)
SecureSSD significantly reduces performance overhead of data sanitization (11% slowdown at most)
Results: Lifetime

Write Amplification Factor (WAF) = \( \frac{\text{# of logical pages written by the host system}}{\text{# of physical pages written by the SSD}} \)

No additional copy in SecureSSD: No lifetime overhead
Results: Effect of Selective Data Sanitization

Selective data sanitization minimizes performance overheads (6% slowdown at most with 60% security-sensitive data)
Other Analyses in the Paper

- Empirical Study on Invalid Data in SSDs
- Reliability Issues in Physical Data Destruction
- Design Space Exploration for pLock and bLock
- Effectiveness of bLock command
Secure Deletion in NAND Flash-Based SSDs

Evanesco: Lock-Based Data Sanitization
- pageLock: Page-Level Data Sanitization
- blockLock: Block-Level Data Sanitization
- SecureSSD: An Evanesco-Enabled SSD

Evaluation

Conclusion
Conclusion

- **Challenges of data sanitization** in NAND flash-based SSDs:
  - *Erase-before-write property →* no overwrite on stored data
  - *Physical data destruction →* high performance & reliability overheads

- **Evanesco**: Uses on-chip access-control mechanisms
  - **pLock**: Page-level data sanitization
    - Implements the access-permission flag of each page using spare cells
  - **bLock**: Block-level data sanitization
    - Programs the SSL of a block to disconnect all pages
  - **SecureSSD**: An Evanesco-Enabled SSD
    - Supports selective data sanitization to reduce performance overheads

- **Results**
  - Provides the same level of reliability of an unmodified SSD
    - Validated w/ 160 real state-of-the-art 3D NAND flash chips
  - Significantly improves performance and lifetime over existing data-sanitization techniques
    - Provides comparable (94.5%) performance with an unmodified SSD
Evanesco: Architectural Support for Efficient Data Sanitization in Modern Flash-Based Storage Systems

Myungsk Kim*, Jisung Park*, Geonhee Cho, Yoona Kim, Lois Orosa, Onur Mutlu, and Jihong Kim

Seoul National University
SAFARI Research Group, ETH Zürich

ASPLOS 2020

*M. Kim and J. Park equally contributed.