Achieving Fast Sanitization with Zero Live Data Copy for MLC Flash Memory

Ping-Hsien Lin[1,5], Yu-Ming Chang[2,6], Yung-Chun Li[1,7], Wei-Chen Wang[1,2,8], Chien-Chung Ho[3,9], Yuan-Hao Chang[4,10]
[1]Macronix International Co., Ltd., Emerging System Lab., [2]Dept. of CSIE, National Taiwan University,
[3]Dept. of CSIE, National Chung Cheng University, [4]Institute of Information Science, Academia Sinica
[5]pinghsienlin@mxic.com.tw, [6]sanforever.chang@gmail.com, {[7]monixslee, [8]raymondwang}@mxic.com.tw,
[9]ccho@cs.ccu.edu.tw, [10]johnson@iis.sinica.edu.tw

ABSTRACT

As data security has become the major concern in modern storage systems with low-cost multi-level-cell (MLC) flash memories, it is not trivial to realize data sanitization in such a system. Even though some existing works employ the encryption or the built-in erase to achieve this requirement, they still suffer the risk of being deciphered or the issue of performance degradation. In contrast to the existing work, a fast sanitization scheme is proposed to provide the highest degree of security for data sanitization; that is, every old version of data could be immediately sanitized with zero live-data-copy overhead once the new version of data is created/written. In particular, this scheme further considers the reliability issue of MLC flash memories; the proposed scheme includes a one-shot sanitization design to minimize the disturbance during data sanitization. The feasibility and the capability of the proposed scheme were evaluated through extensive experiments based on real flash chips. The results demonstrate that this scheme can achieve the data sanitization with zero live-data-copy, where performance overhead is less than 1%.

KEYWORDS

Sanitization, flash memory, page programming

1 INTRODUCTION

The explosive growth of data-driven applications (e.g., big data) leads to fast-increasing demands on high-speed and low-cost storage devices. Among them, multi-level-cell (MLC) flash chips have become the most pervasive media in modern storage systems due to its low bit-cost. Unlike DRAM or hard disks, NAND flash chips do not support in-place updates, so that it is hard for flash storage devices to ensure data security. For example, traditional approaches, such as overwriting all-zero data, for data deletion cannot be directly applied to flash storage devices. This is because the write constraint of flash memory will result in multiple versions of data for the same logical address to co-exist in flash memory when using the existing management designs for flash memories. Once the flash storage device is used to store confidential data, those old versions of data expose a high risk of being stolen by malicious attackers if a flash chip is de-soldered out of the storage device. Therefore, the concept of "sanitization" was proposed to completely remove or destroy the data, including old versions, from the storage device permanently. Nevertheless, it is not easy to realize the data sanitization with an efficient design in flash storage devices due to the write constraint of flash memories. Meanwhile, it would be more challenging to manage or delete multiple versions of data in MLC flash memories, in which the disturbance issue is more serious and the page programming has more restrictions. These observations motivate us to seek for a cost-effective solution to delete every old version of data any time for MLC flash-based storage devices.

A NAND flash chip usually includes one or more planes, and each plane contains a plurality of blocks. A block comprises a fixed number of pages, where a block is the minimum unit for erase and a page is the basic unit for read and write. A page is composed of a large number of cells, and these cells can be further divided into a data area and a spare area (known as out-of-band). The data area is used to store user data, while the spare area is used to store some house-keeping information, e.g., the error correction code (ECC) [10, 12] and mapping information. Flash chips have a special write constraint called *write-once constraint*. That is, a flash page that has been written cannot be updated directly until its residing block is erased first. As a result, updated data are typically written to a free page. This is called *out-place-update*, which typically relies on a layer, called *flash translation layer (FTL)*, to handle the address translation. The aim of address translation is to map the logical block address (LBA) of a read/write request from the host to the corresponding physical address/page in a flash memory chip. A page that contains the latest (resp. old) version of data is called a valid (resp. invalid) page. As the host keeps writing data, the number of free pages will decrease as well. Once the number of free pages drops below a predefined threshold, *garbage collection (GC)* [19] is invoked to reclaim those invalid pages containing out-of-date data. In general, garbage collection usually involves some read or write operations for live-data-copy to backup live pages and some erase operations for resetting blocks. However, a flash block typically has a limited number or program/erase (P/E) cycles, e.g., 100K for SLC, 10K for MLC, and 1K–3K for TLC flash chips. Therefore, wear-leveling [6] techniques are needed to distribute erase operations over entire flash space as evenly as possible.

In contrast to traditional hard disks where each sector can be overwritten with all-zero data to realize the security erase commands for ATA and SCSI protocols [1, 2], flash memories cannot

ICCAD '18, November 5–8, 2018, San Diego, CA, USA

ACM ISBN 978-1-4503-5950-4/18/11.
https://doi.org/10.1145/3240765.3240773

support such a feature of "in-place-update." To meet the requirement of data sanitization on flash-based storage systems, some excellent works have been proposed and can be categorized into two types: encryption-based and erase-based approaches. The main idea of encryption-based approaches [5, 15] is to erase only the key instead of data such that those versions of data encrypted by the same key cannot be recovered. However, the key management is non-trivial and the encrypted data are possibly deciphered before an actual erase is issued to the residing block. The erase-based approaches [11] aim at destroying data physically and timely from the flash memories. These approaches will incur significant overhead on copying live data before their residing blocks are erased. However, very little work has been done to resolve this issue from the "programming" perspective of flash cells/pages. The most closely related work [20], called *scrubbing*, is to rewrite the page with all-zero data with the same page program command provided by the flash memory. However, if the scrubbing technique is directly applied to MLC flash memories, it is inevitable to do live-data-copy because the data on a valid upper page or a lower page would be destroyed. Moreover, the scrubbing technique will introduce a serious disturbance problem on performing sanitization. Such a reliability problem will be further worsened in 3D NAND flash memories [7, 8].

In this work, a fast sanitization scheme is proposed to realize the data sanitization requirement for MLC flash-based storage systems. In order to achieve the highest degree of security, the proposed scheme is able to remove the old version of data immediately after a new version of data is written. We must point out that the proposed scheme is orthogonal to the previous work in encryption-based or erase-based approaches. Our scheme can completely remove the needs of live-data-copy (i.e., live page copy) to achieve the data sanitization by means of skillful programming. In addition, to consider the reliability issue of MLC flash memories, one-shot sanitization design is proposed to sanitize data using only one program voltage. As a result, the disturbances to both the same page and other pages are minimized and the performance is optimal for the sanitization with a programming-based style. The evaluation done in real flash chips demonstrates the feasibility of the proposed scheme. The results show that the proposed scheme only has less than 1% performance overhead, as compared to the system without supporting data sanitization.

The rest of this paper is organized as follows. Section 2 presents the background knowledge and our research motivation. Section 3 presents the proposed scheme for data sanitization in detail. In section 4, we evaluate the proposed scheme through extensive experiments with real flash chips. Section 5 concludes this work.

2 BACKGROUND AND MOTIVATION

2.1 Background Knowledge

In recent years, NAND flash memory has been extensively adopted in commodity storage products, such as Embedded Multi Media Card (eMMC) and solid-state drive (SSD). In order to further reduce the cost, most of the companies of flash products turn to choose multi-level-cell (MLC) or triple-level-cell (TLC) flash memories as the primary storage media. Figure 1 illustrates the structure of the MLC flash memory. A memory cell, marked as "Cell" in the figure, is able to store a number of electrons with its conductive layer (resp. non-conductive layer), which is also called floating-gate (resp. SONOS [14]) device. The number of electrons in this layer determines the threshold voltage (V_t). For an MLC flash memory cell, it has four states, "11", "10", "00" and "01", to represent two-bit data [1]. For example, if a memory cell stores enough electrons and its V_t falls into *window 4*, this cell stores the value "01". On the contrary, if a memory cell after an erase operation, which removes most of the electrons from the cell, it will be at the lowest threshold voltage V_t (i.e., *window 1*), which represents value "11".

A basic read or write command on flash memories is typically operated in the unit of page. In the example of Figure 1, a word line is composed of two pages, namely *upper page* and *lower page*, and each page contains M bits since each cell is able to store two-bit information. As you can notice that, the bits with the same offset in the upper page and the lower page exactly share the same (physical) cell. In general, the most significant bit value (resp. the least significant bit value), abbreviated as MSB (resp. LSB), of a cell indicates the bit value stored in the upper page (resp. lower page). For example, a memory cell storing value "01" means that the corresponding bit position in the upper page (resp. lower page) stores 0 (resp. 1). Therefore, if this memory cell with value "01" losses some electrons due to retention problem and shifts to value "00", then an error bit occurs in the lower page because the value changes from 1 to 0 at the LSB position.

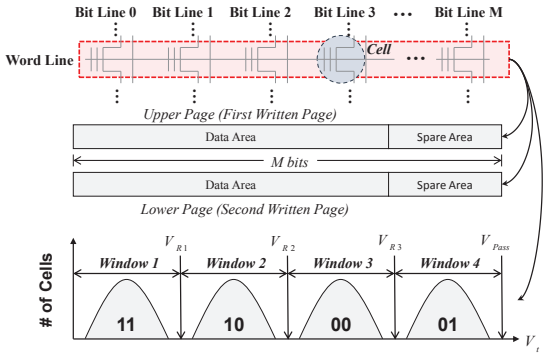


Figure 1: The structure of MLC flash memories.

Owing to the encoding scheme for the four states in the MLC flash memory, reading an upper page or a lower page will have a different operation. The MSB values of windows 1 to 4 are 1, 1, 0, and 0 respectively. Hence, applying one only voltage, i.e., V_{R2}, to the word line is able to separate 0 from 1 to read out the data stored in the upper page. This is because, for those cells storing "11" and "10", voltage V_{R2} can turn them on and to let the current flow through the corresponding bit lines. When the V_t of states "00" and "01" are higher than V_{R2}, their memory cells are turned off and no current is sensed in the page buffer. Similarly, applying two voltages, i.e., V_{R1} and V_{R3}, can read out the data of the lower page because the LSB values of windows 1 to 4 are 1, 0, 0, and 1 respectively. In general, reading a lower page has a higher latency than that of reading an upper page.

[1]For an SLC flash memory cell, there are two states "1" and "0" used to store one-bit data.

2.2 Research Motivation

In a typical design management of flash memories, *out-of-place-update* is the most commonly used strategy since a written flash page cannot be overwritten until its residing block is erased. As a result, there will be many versions of data left among the entire flash space as the time goes by. When the security is a primary consideration, multiple versions of sensitive data will expose to a high-risk environment. *In order to have a higher degree of security, it is necessary to delete the old versions of sensitive data as soon as possible after new data is written.* Intuitive designs to fulfill this requirement is to use block erase commands to remove each old version of data as shown in the Figure 2. The first design example in the left part of the figure is *in-place-update*. Each update to a logical address "A" requires several page read/write commands in order to move out the data of valid pages in the same block. Then, a time-consuming erase command is used to reset the block that contains the old version of LBA "A". This design is infeasible since it not only significantly degrades the performance but also harms the flash lifetime because of the excessive erase commands used to destroy (or sanitize) old versions of data. The second design example in the right part of Figure 2 is to leverage the hint from host. That is, the old versions of those sensitive data are deleted only when the host actively requests to delete one LBA, e.g., "A" (Step 3). However, this design imposes a significant overhead on FTL management that needs to track all versions of each LBA. Moreover, the degree of security is sacrificed because those old versions of data are possibly stolen before the deletion indication is issued by the host.

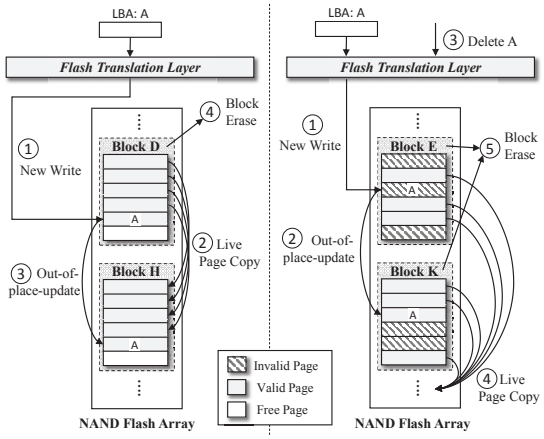


Figure 2: Two design examples of destroying/sanitizing the old version of data for a logical block address (LBA) A.

To mitigate the additional overhead, e.g., live-data-copy, on sanitizing old versions of data, one research work [20] proposed to destroy the data on a flash page by rewriting the page with all zero data, called *scrubbing*. However, this approach was proposed for SLC flash memory and cannot be directly applied to MLC flash memory. The upper and lower pages of each pair in MLC flash memory share the same set of memory cells; as a consequence, scrubbing any page will wipe the data stored in the page of the same pair. In the end, it is inevitable yet necessary to copy the MLC pages with valid data to a free space before deleting (or scrubbing) the other paired MLC pages. In addition, such a rewriting will introduce serious disturbance to other pages of the same block. Thus, more pages are possibly disturbed and need to be copied to other blocks.

The motivation of this work is that we are eager to seek for a low-cost scheme to efficiently sanitize data over MLC flash memories. In particular, we pursue the highest degree of security; that is, every old version of data shall be destroyed/sanitized whenever new data arrives. The objective is to minimize the performance overhead on destroying old data versions and to minimize the side effect [2] on the data that is in the same block but is not required to be deleted. Thus, the design challenges will be on (1) how to realize destroying data of any specified page in MLC flash memories without any live page/data copying, (2) how to minimize the disturbance on the pages that are in the same block but are not required to be deleted, and (3) how to support the new low-cost scheme in the FTL management with minimized overhead.

3 A FAST SANITIZATION SCHEME FOR MLC FLASH MEMORY

3.1 Overview

In this section, a fast sanitization scheme is proposed to securely delete (or sanitize) the old version of data very efficiently. Hereafter, we refer to *delete* and *sanitize* interchangeably in this work. This scheme totally removes the needs of live page copy and to minimize the disturbance on any unrelated pages in MLC flash memories. The scheme provides the highest degree of security without active deletion indications from the host; that is, only the latest version of data of each LBA is kept in the flash storage device at any moment. Hence, no additional overhead of mapping maintenance is required to track all versions of each LBA.

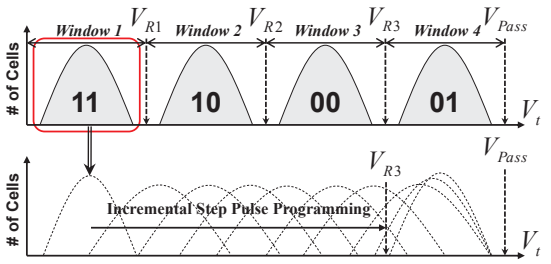
In the following sections, we will first introduce the design concept of how to sanitize an upper or lower page for MLC flash memories in Section 3.2.1. Then, we will discuss how to sanitize the MLC pages in different orders, e.g., lower page first or upper page first, and how to eliminate the disturbance with an *one-shot sanitization design* during the sanitization in Section 3.2.2. Finally, an example of system architecture with the proposed scheme is presented in Section 3.3 to demonstrate how the proposed design can be integrated into the management of MLC flash memories. Note that, due to a relatively low design complexity, the proposed fast sanitization scheme could be applied to MLC flash memories without (or with limited) modifications to the existing designs in flash translation layers.

3.2 A Fast Sanitization Scheme

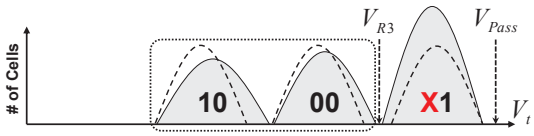
3.2.1 Design Concept. The proposed design aims at sanitizing an upper (resp. lower) page without destroying the data in their associated lower (resp. upper) page, so that there is no need to back up one MLC page before data sanitization is performed. The basic idea is to only reprogram a portion of cells to overlap the cells of other windows by incremental step pulse programming (ISPP) [17]; in this way, the data in the specified upper (or lower) page will be

[2] In contrast to SLC flash memory, simply re-programming MLC pages to destroy data will have serious disturbance to its sibling pages in the same block.

destroyed and cannot be read back with any read voltage. Figure 3 is an example to illustrate the main idea for sanitizing an upper page. Before sanitization, there are four possible states, "11", "10", "00" and "01", for each memory cell as shown in the top of Figure 3(a). The MSB (resp. LSB) value of each state in a memory cell indicates the bit value stored in the corresponding bit position of an upper (resp. lower) page.



(a) Programming "11" cells from window 1 to window 4 using incremental step pulse porgramming (ISPP).



(b) The final result of V_t distribution after the sanitization.

Figure 3: Design concept of sanitizing the data in an upper page by incremental step pulse programming (ISPP).

Our goal is to destroy the data on the upper page and the validity of corresponding lower page is preserved in this example. What we shall care about is the MSB value of memory cells' states. Therefore, the idea is to use ISPP to program those "11" cells to the highest V_t at window 4, which originally contains "01" cells (see Figure 3(a)). This will lead to the overlap of "11" and "01" cells, and thus the MSB values stored for the upper page are ruined, where the ruined bit is marked as "X" in Figure 3(b). Note that ISPP is used to make sure the V_t of all "11" cells have to pass the target voltage, i.e., V_{R3} , and a read voltage (V_{R1}) is used to identify the cells at window 1 before the ISPP operation. Otherwise, some of "11" cells may fall into the range of window 3 to ruin the LSB values of the lower page. Since the memory cells storing 0 and 1 for the same upper page are programmed to a similar V_t in the same window, no read voltage (including V_{R2}) can be used to identify the value stored (in memory cells) for the upper page.

After sanitization, the data in the lower page are still readable and there is no difference to read a lower page by applying the same two read voltages, i.e., V_{R1} and V_{R3}, to the selected word line. This is because all the memory cells storing value 0 for a lower page are still in the V_t range between V_{R1} and V_{R3}, and the memory cells storing value 1 are at the V_t range larger than V_{R3}. Therefore, the proposed scheme can avoid the needs on copying or back up the data in the lower page before sanitization. Astute readers might point out that one read voltage V_{R3} is enough to read out the data in the lower page with a better performance. Nevertheless, to simplify the read circuit at the hardware layer and to avoid the maintenance of read voltage information at the software layer, we favor using the same logic and voltages, i.e., V_{R1} and V_{R3}, to read a lower page without considering whether this page is before or after sanitization.

Even though this design concept seems valid to sanitize an upper or a lower page, the adopted ISPP still have some implementation challenges with consideration of the performance and reliability. The first problem is the performance of sanitization. In typical, ISPP repeatedly executes cycles composed of a program operation and a verify operation to push memory cells forward until all cells have reached the target V_t. This procedure is long-winded and will make the sanitization latency as long as writing a page. The second problem is the reliability caused by ISPP. Programming "11" cells from window 1 to window 4 will go through the longest path of V_t change. As a result, more cycles are required in ISPP and will inevitably introduce serious disturbance, especially for MLC flash memories, to not only other cells in the same page but also the adjacent pages in the same block.

3.2.2 One-shot Sanitization Design. In order to achieve a better performance and reliability based on the proposed design concept in our scheme, *one-shot sanitization design* is presented in this section. This design is called *one-shot programming*. Its objective is to extremely reduce the number of cycles required by ISPP to only one cycle and to minimize the change of V_t for those memory cells to-be-sanitized. Consequently, the latency of sanitization is reduced and the disturbance to other unrelated cells of the same page or the adjacent pages of the same block is mitigated. The illustration of the one-shot sanitization design is divided into three sanitization scenarios. The first scenario is to sanitize the upper page and then the lower page. On the contrary, the second scenario is that the lower page is sanitized before the upper page is sanitized. The last scenario is to sanitize both upper and lower pages at the same time. Note that, the upper page and the lower page in all scenarios are in the same set of memory cells.

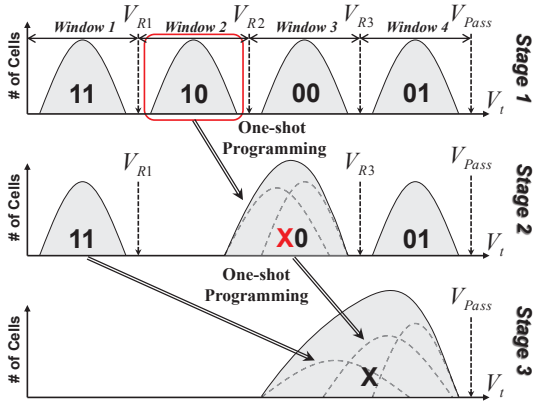


Figure 4: The design of sanitizing an upper page and then a lower page.

Figure 4 illustrates the first scenario for sanitizing the upper page and then the lower page. Similarly, there are four states in the beginning (see Stage 1). To sanitize the upper page, the "10" cells in window 2 is programmed forward to overlap the range of window 3 containing "00" cells using one-shot programming. In such a way, the MSB values of upper pages are destroyed and marked as "X" in Stage 2 of Figure 4. The philosophy behind this design is two-fold. First, instead of programming "11" cells to window 4 in

the previous example (see Figure 3), this design choice minimizes the required change of V_t for those memory cells that need to be programmed for sanitizing the upper page. Second, one-shot programming can be used to greatly reduce the time on sanitization. This is because windows 2 and 3 are adjacent. That is, the "10" cells are programmed forward with one only programming voltage and do not need a verify operation (or a verify voltage) to make sure whether they all pass a specific V_t value [3]. Note that, the maximum voltage used for one-shot programming is predefined by profiling [4] before the shipment of flash chips. The only restriction is that the V_t of all "10" cells in window 2 cannot exceeds V_{R3} after the programming.

After sanitizing the data on the upper page in Stage 2, we are still able to read out the data of the lower page by applying two voltages, i.e., V_{R1} and V_{R3}. Next, if the data of the lower page is required to be sanitized as well, the simplest way is to program all cells using one-shot programming. Since all memory cells contain don't-care values, a large program voltage can push all cells forward and let all cells get a similar value of V_t. As a result, the LSB value of the lower pages are destroyed as shown in Stage 3 of Figure 4.

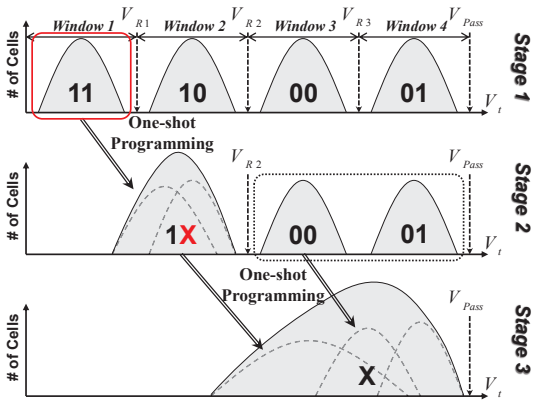


Figure 5: The design of sanitizing a lower page and then an upper page.

The second scenario is to sanitize the lower page first before sanitizing the upper page as illustrated in Figure 5. Likewise, the objective is to minimize the change of V_t value during sanitization. Thus, programming "11" cells to overlap the range of window 2 which contains "10" cells is the best alternative to destroy the LSB value for the lower page (see Stage 2 of Figure 5). Meanwhile, one-shot programming can be used again to achieve the best sanitization performance due to the same reason explained previously for the first scenario. There is one possible question why we did not choose to program "00" cells forward to overlap the range of window 4 for the sanitization. The reason is also two-fold. First, selecting "11" cells for the programming only needs one voltage, i.e., V_{R1}; however, selecting "00" cells shall require two voltages, i.e., V_{R2} and V_{R3}, to identify the "00" cells. Second, programming a "00" cell, which has a higher V_t, will require a higher program voltage than programming a "11" cell. Furthermore, a higher program voltage will cause a bigger disturbance to other pages. After sanitizing the data on the lower page, reading the upper page can be achieved by applying one voltage V_{R2}. If the data of the upper page is required to be sanitized further, we could also program all cells using one-shot programming to destroy the remaining MSB values of the upper page as shown in Stage 3 of Figure 5.

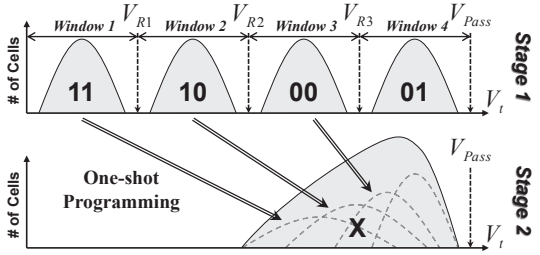


Figure 6: The design of sanitizing both high and low pages at the same time.

The last scenario is to sanitize both the upper and lower pages at the same time as illustrated in Figure 6. This scenario is required when several pages that need to be sanitized and are adjacent. One big advantage is that we could double the speed of sanitization. This is because the data in both upper and lower pages are all invalid such that one-shot programming could program all cells to the similar V_t to destroy all MSB and LSB values at the same time.

3.3 One System Architecture Example

In this section, an example system architecture with the fast sanitization scheme is proposed as shown in the left part of Figure 7. To realize the proposed scheme, the flash chip needs to support three new commands, i.e., *upper-first sanitization*, *lower-first sanitization*, and *all sanitization*, for the three sanitization scenarios. The upper-first (resp. lower-first) sanitization command is used to sanitize the upper (resp. lower) page when its associated lower (resp. upper) page is still valid as shown in Stages 1-2 of Figure 4 (resp. Figure 5). The all sanitization command is to sanitize both upper and lower pages using one-shot programming such as Stages 2-3 of Figure 4, Stages 2-3 of Figure 5, and Stages 1-2 of Figure 6. To support the new commands, three related drivers shall be implemented in the memory technology device layer (MTD), which is a hardware abstraction layer for providing the primitive functions, including the fundamental page read and write, of a flash chip.

Our objective is to provide the highest degree of security; in other words, there is at most one version of data for each LBA among the entire flash storage space. A module, called *Sanitizer*, is implemented at the flash translation layer (FTL) to help sanitize the old version of data whenever a write request is issued from the host. The right part of Figure 7 illustrates the flow chart used in the sanitizer. The write requests from the host can be divided into two categories. The first category writes an LBA, to which no data has been written; namely, this is the first write to the LBA and there is no old version of that LBA in the flash space. As a result, we could directly program the data of this LBA to a free page according to the free space allocator used in the existing FTL. The

[3] In contrast to the example in Figure 3, all "11" cells that need to be programmed must pass the voltage V_{R3} using ISPP during sanitization.
[4] The profiling is to measure the maximum value of voltage that can be used to program the rightmost "10" cells, and the final values of V_t of these cells cannot exceed V_{R3}.

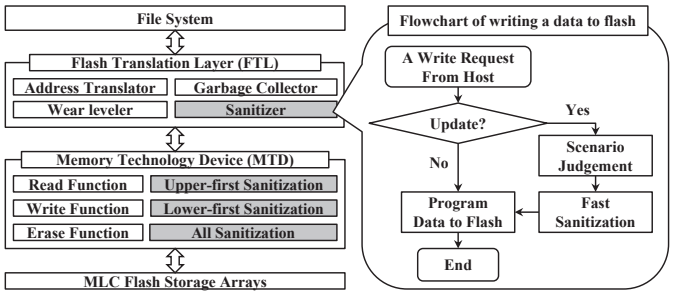


Figure 7: System architecture for the fast sanitization scheme.

second category is to update the old data of an LBA. Note that, both categories can be easily identified by reading the mapping table of FTL. If this case is an update to an LBA, then it is necessary to first sanitize the old version of the data (i.e., the old data version) for the LBA. The physical page address of the old data version can be found from the mapping table and thus the next step is to check whether the to-be-sanitized page is an upper or lower page (see the block "Scenario Judgement" in the flowchart). Then, the corresponding sanitization function, e.g., upper-first and lower-first sanitization, at the MTD layer is invoked to sanitize this page based on the scenario discussed in Section 3.2.2. Finally, it is time to program the new data for the LBA. It is worthy of noting that the design logic for the sanitizer is very simple yet effective and beneficial to integrate the proposed scheme into many existing FTLs with minimized modification efforts.

4 PERFORMANCE EVALUATION

4.1 Experimental Setup

In this section, a series of experiments of device-level and system-level were conducted to evaluate the feasibility and capability of the proposed fast sanitization scheme. For the evaluation at device-level, the proposed one-shot sanitization design was implemented in our MLC flash chips based on an in-house development platform [9] as shown in Figure 8. The flash chip for evaluation was put on a device-under-testing (DUT) board, and the platform is able to do a fine-grained adjustment for each pin of the flash chip in an engineering mode. All three commands, i.e., *upper-first sanitization*, *lower-first sanitization*, and *all sanitization*, for all scenarios are built in this chip, and their corresponding timing delay could be measured by the platform. In addition, the rule of Z-shape programming sequence [16] [5] in typical MLC flash chips is also followed. The objective of the experiment at device-level is to evaluate the feasibility of our one-shot sanitization design, and thus the minimal requirement is to make sure all errors are correctable after using one-shot sanitization with the same ECC capability specified in the original datasheet.

For the evaluation at system-level, three schemes, i.e., *no-security*, *fast-sanitization*, and *scrubbing*, were implemented and compared. In no-security scheme, the system does not pay additional overhead to sanitize the old version of data for each write; as a result, this scheme theoretically has the best performance. Fast-sanitization

[5] Z-shape programming sequence is the most commonly-used strategy in modern MLC or TLC flash chips to alleviate the impact of program disturbance.

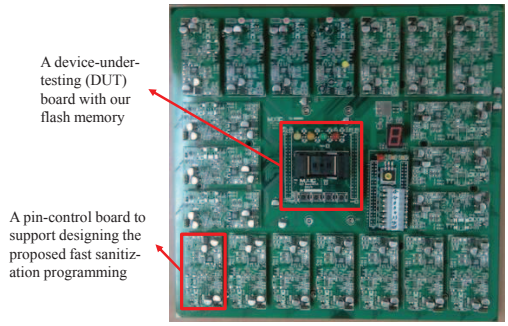
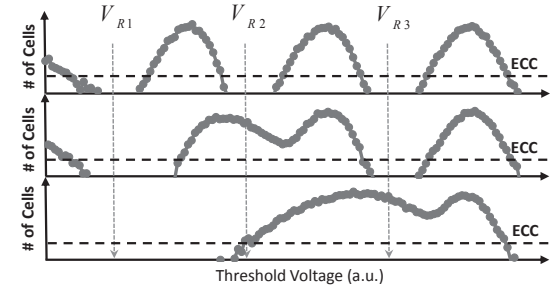


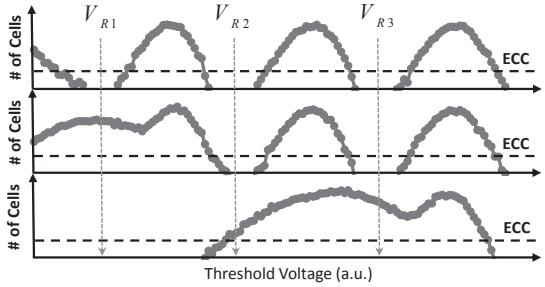
Figure 8: The evaluation platform is used to measure the timing delay of all sanitization commands.

scheme is our proposal and the scrubbing scheme is to rewrite all-zero data to the page that needs to be sanitized [20]. All the investigated schemes adopt the most representative FTL [4], i.e., page-level mapping, implemented in our system. Our objective is to avoid additional live page copies and costly erase operations as much as possible; therefore, all schemes were evaluated in terms of performance and reliability. The performance was evaluated based on the average write latency, the average read/write latency, the number of live-data-copies, and write amplification. The reliability was evaluated based on the number of erase operations issued to the flash chips. To investigate the results under different benchmarks, the experiment was evaluated through real traces. These are public traces obtained from SNIA IOTTA Repository [3], postmark [13], and Microsoft Research Cambridge [18].

4.2 Experimental Results



(a) Sanitize the upper page first, and then the lower page.



(b) Sanitize the lower page first, and then the upper page.

Figure 9: The results of using one-shot sanitization.

4.2.1 Device-level Results. Figure 9 shows the results of V_t distribution when one-shot sanitization design of the proposed scheme is used for all discussed scenarios, where the x-axis denotes the threshold voltage V_t [6] and the y-axis denotes the number of cells. For the results in Figure 9(a) and Figure 9(b), all data that do not need to be sanitized are all readable after ECC at all stages. These results prove the feasibility of the proposed one-shot sanitization design. This is because our design only requires single programming for each sanitization and thus theoretically has the minimal disturbance to other pages in the same block during the sanitization.

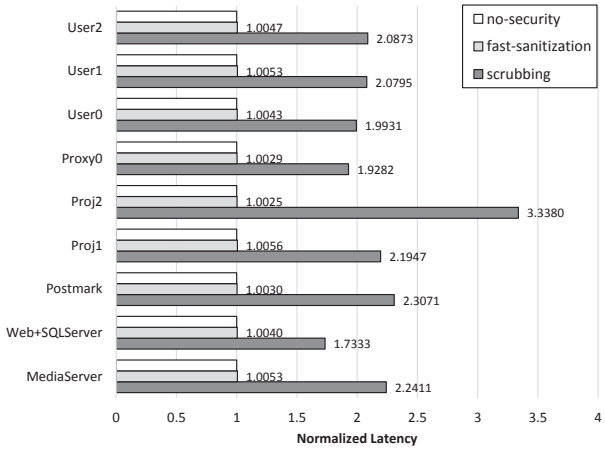


Figure 10: The results of average write latency for all schemes under different workloads.

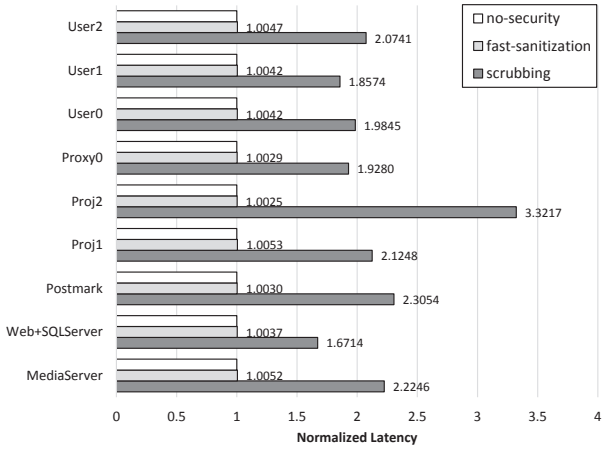


Figure 11: The results of average read/write latency for all schemes under different workloads.

4.2.2 System-level Results. Figure 10 and Figure 11 show the results of average write latency and average read/write latency respectively with different schemes, where the x-axis donates the latency normalized to the results of no-security scheme and the y-axis denotes the different workloads. In general, it is observed that the latency of our proposed scheme ranges from 1.0025 to 1.0056, which is very close to the results of no-security scheme that theoretically has the best performance. The reason is that the proposed one-sanitization design for different scenarios has very little latency overhead on sanitizing an MLC page. On the other hand, the scrubbing scheme has a longer latency ranging from 1.7333 to 3.338. The reason is that it requires to actively copy/duplicate the data for the associated MLC page before rewriting all-zero data to the to-be-sanitized MLC page. Moreover, the additional live-data-copies might introduce more erase operations during sanitization, and thus the latency on serving a write request is further lengthened. These results demonstrate the proposed scheme only introduces less than 1% latency in terms of time overhead while providing the highest degree of security at the same time.

[6] A.U. means any unit.

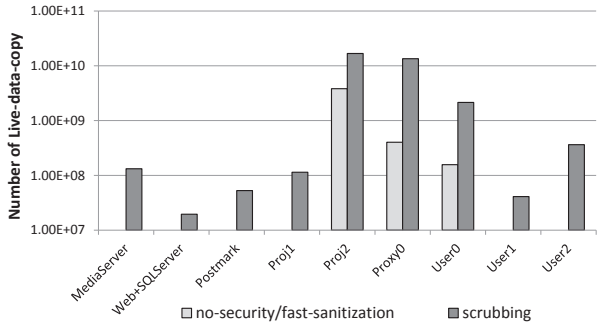


Figure 12: The results of the number of live-data-copies for all schemes under different workloads.

Figure 12 shows the results of the total number of live-data-copies for all the investigated schemes, where the y-axis denotes the value in the "page" unit. This result includes the number of live-data-copies required by garbage collection and by sanitization. As expected, the results of the proposed scheme and the no-security scheme are the same. This is because there is no additional live-data-copy required by the proposed scheme for MLC flash memories. It is found that a larger number of live-data-copies for the scrubbing scheme is required. The reason is also due to the overhead for the duplication/backup of old versions of data before sanitizing an MLC page. This also concurs with results shown in the Figure 10 and Figure 11. For example, under workload proj2, the scrubbing scheme has the largest number of live-data-copies and thus has the longest latency accordingly.

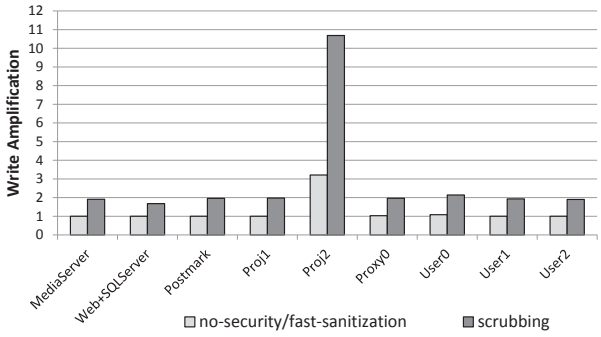


Figure 13: The results of write amplification for all schemes under different workloads.

Figure 13 shows the results of write amplification for all schemes, where the write amplification denoted in y-axis is ratio of the amount of data written to the flash memory to the amount of data written by the host.This is also expected that the proposed scheme and no-security scheme have the same results because our scheme does not need additional live-data-copy for the sanitization. In general, the write amplification of the scrubbing scheme is two times that of the proposed scheme. This is because, for the scrubbing scheme, it typically needs to back up the data of the upper or the lower pages for the sanitization for each update to an LBA. In addition, we observed an interesting exception that, for proj2, the result under the scrubbing scheme has a worse value greater than two times that of the result under the proposed scheme. This is because the additional writes for the backup in the scrubbing scheme further introduce more garbage collection, which comprises many extra reads, writes, and erases.

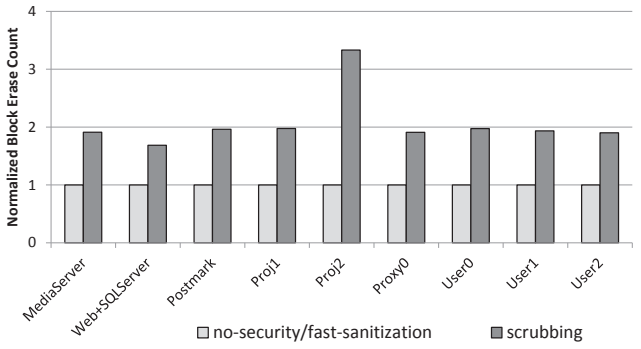


Figure 14: The results of normalized number of block earse counts for all schemes under different workloads.

Figure 14 shows the results of the block erase count for all schemes, where the y-axis denotes the number of block erase count normalized to the results of no-security scheme. Based on the observed results, the scrubbing scheme will lead to more erase operations. The reason is that the scrubbing scheme has a relatively high value of write amplification shown in Figure 13. Those additional writes for backup during sanitization typically contain cold data; meanwhile, they have less chances to be invalidated after being written to a free space. Even worse is that these writes will lead to frequent garbage collection, which introduce more erase operations. In the proposed fast-sanitization scheme, there is no need for these backups and shall have no overhead from the additional erases; this is the same as the no-security scheme. As a result, the proposed scheme is able to provide the security without scarifying the reliability.

5 CONCLUSION

To meet the data security requirement for MLC flash-based storage devices, a fast sanitization scheme is proposed to sanitize the data with zero live data copy. To provide the highest degree of security, the proposed scheme is able to remove the old version of data immediately after the new data is written. While the disturbance effect is an key issue in MLC flash memories, the scheme skillfully sanitizes the upper or the lower pages with only one program shot to alleviate the side effect on other pages. Furthermore, this one-shot sanitization can deliver the optimal performance for the programming-based sanitization. The evaluation is conducted at device-level and system-level to demonstrate the feasibility and the capability respectively of the proposed scheme. In device-level evaluation, we successfully implement the one-shot sanitization design and demonstrate that all other data that do not need to be sanitized are not affected by disturbance and are correctable by the same ECC specification. At system-level evaluation, the proposed scheme only incurs less than 1% performance overhead and can totally remove the needs of live-data-copy.

REFERENCES

[1] 1997. American National Standard of Accredited Standards Committee X3T13. *Information Technology - AT Attachment-3 Interface (ATA-3)* (January 1997).

[2] 1998. Technical Committee of Accredited Standards Committee NCITS T13. *Information Technology - AT Attachment with Packet Interface Extension* (August 1998).

[3] 2011. *SNIA IOTTA Repository.* http://iotta.snia.org/.

[4] Amir Ban. 1995. Flash file system. US Patent 5,404,485.

[5] Alexandre Braga and Alfredo Colito. 2014. Adding Secure Deletion to an Encrypted File System on Android Smartphones *(IARIA '14).* 106–110.

[6] Li-Pin Chang. 2007. On efficient Wear Leveling for Large-scale Flash-memory Storage Systems. In *Proc. of the ACM SAC.* 1126–1130.

[7] Yu-Ming Chang, Yuan-Hao Chang, Tei-Wei Kuo, Hsiang-Pang Li, and Yung-Chun Li. 2013. A Disturb-alleviation Scheme for 3D Flash Memory. In *Proc. of the IEEE ICCAD.* 421–428.

[8] Yu-Ming Chang, Yuan-Hao Chang, Tei-Wei Kuo, Yung-Chun Li, and Hsiang-Pang Li. 2016. Disturbance relaxation for 3D Flash memory. *IEEE Trans. Comput.* 65, 5 (2016), 1467–1483.

[9] Yuan-Hao Chang Chien-Chung Ho, Yung-Chun Li and Yu-Ming Chang. 2018. Achieving Defect-Free Multilevel 3D Flash Memory with One-Shot Program Design. In *Proc. of the IEEE/ACM DAC.*

[10] H. Choi, W. Liu, and W. Sung. 2010. VLSI Implementation of BCH Error Correction for Multilevel Cell NAND Flash Memory. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 18, 5 (2010), 843–847.

[11] Sarah Diesburg, Christopher Meyers, Mark Stanovich, Michael Mitchell, Justin Marshall, Julia Gould, An-I Andy Wang, and Geoff Kuenning. 2012. TrueErase: Per-file Secure Deletion for the Storage Data Path. In *Proc. of the ACSAC.* 439–448.

[12] R. Gallager. 1962. Low-density Parity-check Codes. *IRE Transactions on Information Theory* 8, 1 (1962), 21–28.

[13] Jeffrey Katcher. 1997. *Postmark: A New File System Benchmark.* Technical Report. Technical Report TR3022, Network Appliance.

[14] Chang-Hyun Lee, Sung-Hoi Hur, You-Cheol Shin, Jeong-Hyuk Choi, Dong-Gun Park, and Kinam Kim. 2005. Charge-trapping device structure of Si O 2/ Si N/ high-k dielectric Al 2 O 3 for high-density flash memory. *Applied Physics Letters* 86, 15 (2005), 152908.

[15] Joel Reardon, Srdjan Capkun, and David Basin. 2012. Data Node Encrypted File System: Efficient Secure Deletion for Flash Memory. In *Proc. of the USENIX Security.* 333–348.

[16] S. H. Shin, D. K. Shim, J. Y. Jeong, O. S. Kwon, S. Y. Yoon, M. H. Choi, T. Y. Kim, H. W. Park, H. J. Yoon, Y. S. Song, Y. H. Choi, S. W. Shim, Y. L. Ahn, K. T. Park, J. M. Han, K. H. Kyung, and Y. H. Jun. 2012. A New 3-bit Programming Algorithm Using SLC-to-TLC Migration for 8MB/s High Performance TLC NAND Flash memory. In *2012 Symposium on VLSI Circuits (VLSIC).* 132–133.

[17] Kang-Deog Suh, Byung-Hoon Suh, Young-Ho Um, Jin-Ki Kim, Young-Joon Choi, Yong-Nam Koh, Sung-Soo Lee, Suk-Chon Kwon, Byung-Soon Choi, Jin-Sun Yum, Jung-Hyuk Choi, Jang-Rae Kim, and Hyung-Kyu Lim. 1995. A 3.3 V 32 Mb NAND Flash Memory with Incremental Step Pulse Programming Scheme. In *Proc. of the IEEE ISSCC.* 128–129.

[18] Avishay Traeger, Erez Zadok, Nikolai Joukov, and Charles P. Wright. 2008. A Nine Year Study of File System and Storage Benchmarking. *Trans. Storage* 4, 2 (2008), 5:1–5:56.

[19] Che-Wei Tsao, Yuan-Hao Chang, and Ming-Chang Yang. 2013. Performance Enhancement of Garbage Collection for Flash Storage Devices: an Efficient Victim Block Selection Design. In *Proc. of the IEEE/ACM DAC.* 165:1–165:6.

[20] Michael Yung Chung Wei, Laura M. Grupp, Frederick E. Spada, and Steven Swanson. 2011. Reliably Erasing Data from Flash-Based Solid State Drives. In *9th USENIX Conference on File and Storage Technologies.* 105–117.