International Journal of Security and Its Applications
Vol. 7, No. 3, May, 2013

Page Mapping Scheme to Support Secure File Deletion for NAND-based Block Devices

Ilhoon Shin

Seoul National University of Science & Technology
ilhoon.shin@snut.ac.kr

Abstract

As the amount of digitized private data grows, security requirements such as irrecoverable file deletion are gaining importance. Existing secure file deletion tools, which repeatedly over-write the contents to be deleted, do not work properly with NAND-based block devices because these devices perform out-of-place updates on over-write requests, retaining the original data. This paper presents a method for secure file deletion in NAND-based block devices. The presented scheme maintains the over-write count and the physical locations of the original data. If the over-write count exceeds a predefined threshold, the scheme finds the original data that has been invalidated by previous over-write operations and removes the original data using the block erasure operation. The erased data is irrecoverable. Trace-driven simulation shows that the presented scheme reduces the total I/O time up to 104 % under a realistic workload compared to existing file deletion schemes.

Keywords: *secure file deletion, over-write count, page mapping scheme, flash translation layer, NAND flash memory*

1. Introduction

Nowadays, various kinds of private data such as pictures, personal videos, contact information, and social security numbers are digitized and stored in computing devices, leading to increased security concerns. For example, we need to guarantee that sensitive files that we want to delete are completely removed by the deletion operation. However, traditional file systems such as NTFS, FAT, and EXT3 do not actually delete the file contents when performing file deletion. Only the metadata of the file are deleted; the file contents remain unmodified in order to reduce the latency of file deletion. Thus, the file contents can be recovered by reading the device directly even after the file is deleted.

In order to address the problem, secure file deletion tools repeatedly over-write the file contents with randomly generated data to achieve irrecoverable file deletion, which is called secure file deletion [4]. In standard block devices such as hard disks, the original contents are completely eliminated by over-write operations because the new data are over-written in the original location that held the sensitive data.

However, these secure file deletion tools do not work properly in NAND-based block devices such as solid state drives (SSDs), memory cards, and USB memories because NAND flash memory does not directly support the over-write operation. NAND-based block devices emulate the over-write feature with the out-of-place update, which writes new data to a new location. In the out-of-place update, the original page is preserved even after repeated over-write operations; it is merely marked as invalid for subsequent garbage collection. Thus, the deleted data can be recovered by scanning the NAND flash memory directly even after secure file deletion.

The goal of the paper is to support secure file deletion with low overhead for NAND-based block devices. For this purpose, we modify the existing flash translation layer (FTL) firmware so that it erases sensitive data completely on repeated over-write operations. Trace-driven simulation shows that the presented method causes a performance overhead of 8.6 % under a realistic workload.

The rest of the paper is organized as follows. Section 2 explains the characteristics of NAND flash memory and existing FTL schemes. Section 3 describes our work to implement secure file deletion in NAND-based block devices. A performance evaluation of the scheme is presented in Section 4. Finally, Section 5 states our conclusions.

2. Flash translation layer

NAND flash memory is a kind of electrically erasable programmable read-only memory (EEPROM) that consists of blocks and pages. A page is the unit of a read/write operation, similar to a hard disk sector. A block that consists of multiple pages is the unit of an erase operation. NAND flash memory does not support the over-write operation. Once a page is written, it cannot be re-written before the block that the page belongs to is erased. Thus, implementing the in-place update that over-writes new data to the original location as we do in hard disks is almost impossible, because erasing the block that the target page belongs to also erases the valid contents of the other pages. NAND-based block devices therefore perform out-of-place updates, which write new data to a new location (a clean page). The original page is not modified; it is marked as invalid. In the out-of-place update, the physical location of the valid data changes on every write request, and we need to maintain the mapping information between the logical sector number and its physical location. Performing the out-of-place update, while at the same time maintaining the mapping information of the logical sectors, is a key function of FTL.

The existing FTL schemes are classified into page mapping [6], block mapping [7], and hybrid mapping schemes [1, 2] according to the unit of the mapping information. The page mapping scheme [6] maps the location of the logical sectors in a NAND page unit. On write requests, it searches for a clean page, and the new data are written in the found page. The old page is marked as invalid, and the mapping table is updated accordingly. Thus, the mapping table is a one-dimensional array, where the index is the logical page number (LPN) calculated from the logical sector number, and the value is the physical page number.

If at some point the number of clean blocks drops below the threshold, the garbage collection process is initiated. The garbage collection process selects the victim block that has the highest number of invalid pages [5]. The valid pages of the victim block are moved to another clean block, and finally the victim block is erased. Consequently, the sensitive contents are restorable even after repeated over-write requests until the block that the contents belong to is selected as victim and erased by the garbage collection process.

The block mapping scheme [7] maps the location of the logical sectors in a NAND block unit. On write requests, it searches for a clean block and writes the new data to the found block, writing the unmodified pages of the original block together to the new block. The old block is marked as invalid, and the mapping table is updated accordingly. The mapping table of the block mapping scheme is a one-dimensional array, where the index is the logical block number (LBN) calculated from the logical sector number, and the value is the physical block number. We do not need to store the offset inside a block because the page order inside the block is not changed.

In the block mapping scheme, secure file deletion is easily implemented by instantly erasing the old block on over-write requests [3]. The problem is low performance, which stems from the fact that the block mapping scheme moves the entire block even when over-writing a small portion of the block. This excessive copy overhead significantly hurts the overall performance of NAND-based block devices.

Hybrid mapping schemes combine page mapping and block mapping. They operate like block mapping schemes but use several NAND blocks, called log blocks, as write buffers in order to address the excessive copy overhead. The log blocks are managed by the page mapping scheme, and the other blocks, called data blocks, are managed by the block mapping scheme. Hybrid mapping schemes include the block associative sector translation (BAST) scheme [1] and the fully associative sector translation (FAST) scheme [2], named according to the association between the log blocks and the data blocks.

The BAST scheme associates a log block with a data block in a one-to-one manner. On a write request, it searches for the log block associated with the target data block. If there is no associated log block, a clean log block is associated with the target data block. The new data are written to the associated log block, and the old page is invalidated. Meanwhile, if there is no clean log block, the garbage collection process is initiated. The victim log block selected by the garbage collection process is merged with the associated data block and reclaimed to a clean log block by the block erasure operation.

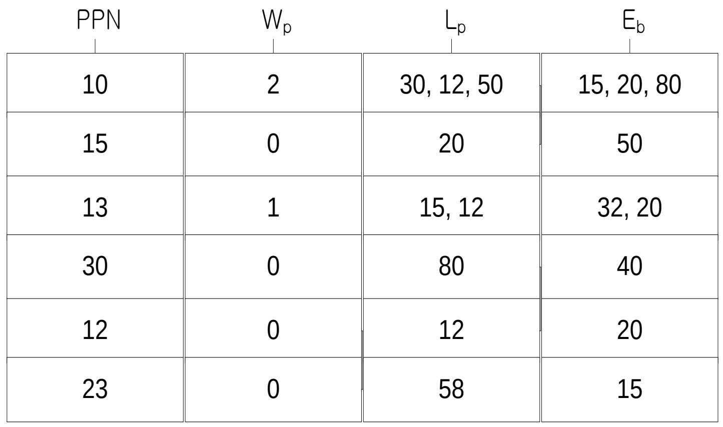
In the BAST scheme, secure file deletion can be guaranteed by repeated over-write requests [3]. The repeated over-write requests exhaust the clean pages of the log block, invoking the garbage collection process. The garbage collection process merges the log block with the associated data block, and all the invalidated pages including the sensitive contents are removed by the block erasure operation. One drawback is low performance for a random write pattern. In the BAST scheme, the log block is associated with one data block, and thus the clean log block is frequently exhausted in a widely distributed random write pattern [2]. The log blocks are frequently merged with the data blocks and remain under-utilized, affecting the overall performance seriously.

In order to solve this problem, the FAST scheme [2] establishes an m-to-n association between the log blocks and the data blocks. On a write request, the new data are written in the current working log block regardless of the target data block. If there is no clean page in the working log block, another clean log block is chosen as the working log block. If there is no clean log block, the victim log block is selected by the First-In First-Out (FIFO) scheme and merged with the associated data blocks. The FAST scheme fully utilizes the log block space even for a random write pattern and delivers better performance than the BAST scheme. However, implementing secure file deletion is not straightforward, because the sensitive contents are restorable until the log block that is associated with the data block containing the sensitive contents is selected as victim and merged with the data blocks [3].

Hence, it is not difficult to implement secure file deletion in the block mapping scheme and in the BAST scheme; the only drawback is poor performance. We describe a method to support high-performance secure file deletion for the page mapping scheme.

3. Secure file deletion for page mapping

In the original page mapping scheme, the exact time at which the deleted data are completely removed by the repeated over-write requests is not guaranteed. The delay can be indefinite if there are sufficient clean blocks. In order to implement secure file deletion, it should be guaranteed that the original physical page will be erased if over-write requests to the same logical page occur at least *n* times. In order to satisfy this condition, we need to maintain the over-write request count (W_p) and the locations of the physical blocks that have the invalidated data (L_p) for each logical page. Thus, the above data (W_p and L_p) are added to the mapping table of the page mapping scheme (Figure 1). In the figure, PPN denotes current physical location (physical page number) of each logical page. Note that L_p is an array that contains all the locations of the physical blocks that have the invalidated data for the logical page. We should maintain all the locations of the invalidated pages, because we do not know which invalidated page holds the original sensitive data.



PPN	W_p	L_p	E_b
10	2	30, 12, 50	15, 20, 80
15	0	20	50
13	1	15, 12	32, 20
30	0	80	40
12	0	12	20
23	0	58	15

Figure 1. Mapping table of the presented scheme

If the number of over-write requests to a logical page exceeds a predefined over-write threshold (UB_w), the secure deletion function is initiated and the blocks that have the invalidated data from the mapping table are found. At this point, we need to check whether the invalidated data are still preserved or have been erased by the previous garbage collection. In case the data were already erased, we do not need to erase them again, because they are already irrecoverable. However, if the data are still preserved, we should reclaim the block with the erasure operation, even if there are still clean pages in the block. Thus, we need to maintain the erase counts (E_b) of the physical blocks of the invalidated data. Like L_p, E_b is also an array and is added to the mapping table (Figure 1). The secure deletion function compares the value of E_b with the current erase counts of the blocks that contain the original pages, and reclaims the block only if the erase counts are the same as the value of E_b. The detailed pseudo code of the presented scheme is shown in Figure 2. We describe only the write function and the secure deletion function of the presented page mapping scheme, because the read function is the same as that of the original page mapping scheme.

```
PageWrite()
{
        get the current working block;
        if (there is no clean page in the block)
        {
                get another clean block or perform the garbage collection;
                make the new clean block the working block;
        }
        write the requested data to the working block;

        increase $W_p$ by 1;
        if ($W_p > UB_w$)
        {
                call SecureDelete ();
        }

        add the number of the working block to $L_p$;
        add the current erase count of the working block to $E_b$;
}

SecureDelete()
{
        foreach (the block listed in $L_p$)
        {
                if (the value of $E_b$ == erase count of the block)
                {
                        move the valid pages of the block to another clean block;
                        erase the block;
                }
        }
        initialize $L_p$ and $E_b$;
        initialize $W_p$ to 0;
}
```

Figure 2. Pseudo code of the presented scheme

4. Performance Evaluation

We compare the performance of the presented scheme with the BAST scheme and the original page mapping scheme through a trace-drive simulation. The block mapping scheme is excluded owing to its considerably poor performance. We use both realistic and synthetic traces. The realistic trace (NTFS) was extracted using the diskmon tool on a Windows PC formatted with the NTFS file system while installing programs, surfing the Internet, editing documents, and so on. The partition size was 32 GB, and the total amount of data written was 35 GB. The synthetic trace (RAND) was generated while issuing random write requests on an 8 GB partition. We assume that the 80% of the total write requests were generated in 20% of the partition. The total amount of data written bytes was 64 GB.

The target NAND-based block device is modeled as consisting of eight NAND flash chips in a 2-channel & 4-way structure. In a 2-channel & 4-way structure, eight pages constitute one clustered page, and are read and written together. Thus, a clustered page is regarded as one physical NAND page by the FTL. Similarly, eight physical blocks constitute one clustered block. A physical page is 2 KB in size and a physical block is 128 KB in size. The read and the write latencies of a physical page are 25 μs and 200 μs, respectively. The erase latency of a physical block is 2 ms. The performance measure is the total I/O time in seconds, calculated using the following formula:

total I/O time = clustered page read count × clustered page read latency + clustered page write count × clustered page write latency + clustered block erase count × clustered block erase latency.

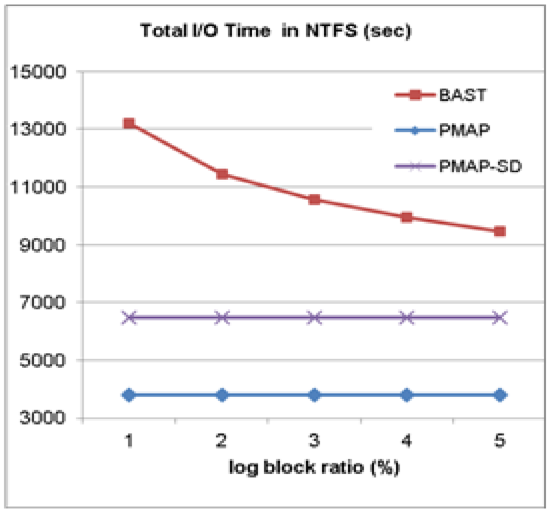


Figure 3. Total I/O time in NTFS

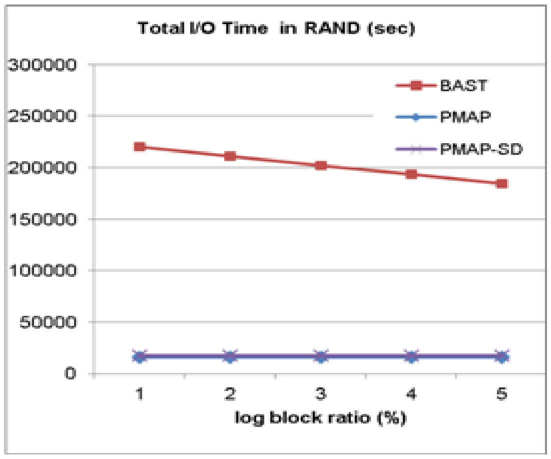


Figure 4. Total I/O time in RAND

Figures 3 and 4 show the performance evaluation results in NTFS and in RAND, respectively. The x-axis denotes the log block ratio of the BAST scheme, which is varied from 1% to 5% of the total NAND blocks. The y-axis denotes the total I/O time in seconds. PMAP and PMAP-SD are the original and modified page mapping scheme, respectively. In the PMAP-SD scheme, UB_w is set to 64 because the BAST scheme uses that value. The figure shows that the presented scheme delivers significantly better performance than the BAST scheme. The total I/O time is reduced from about 46% to 104% in NTFS and 1/12 to 1/10 in RAND. The improvement is much more conspicuous in the RAND trace because the BAST scheme is vulnerable against the random write pattern. Meanwhile, the overhead compared to the original page mapping scheme is about 71% in NTFS and 10% in RAND. In RAND, the overhead is not considerable, because a page is rarely over-written more than 64 times.

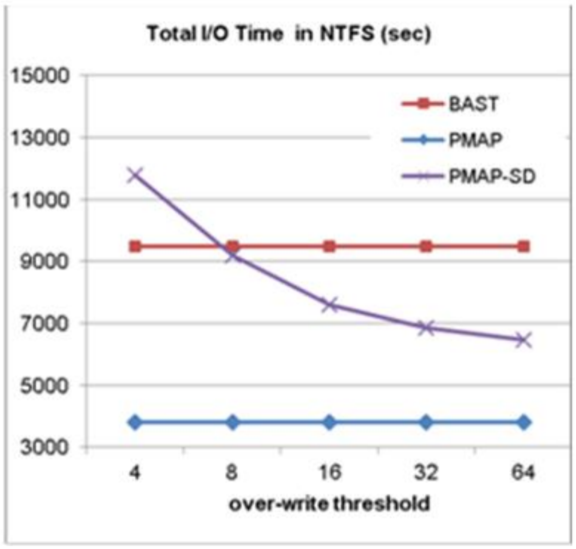


Figure 5. Influence of *UB_w* (the over-write threshold) in NTFS

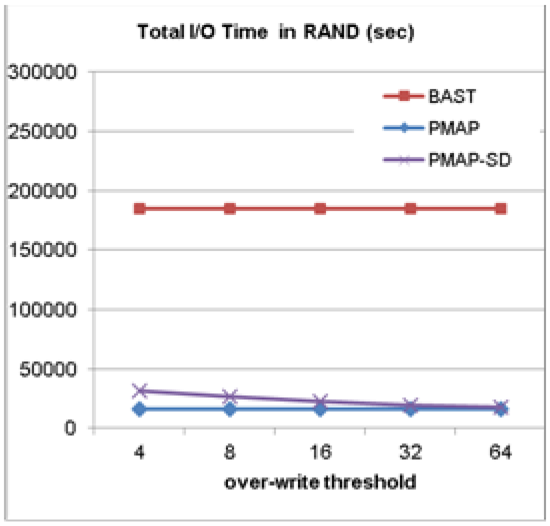


Figure 6. Influence of *UB_w* (the over-write threshold) in RAND

Meanwhile, if we reduce the over-write threshold, UB_w, secure file deletion is more strictly guaranteed. For example, if we set UB_w to 4, the sensitive contents are completely removed if data is over-written more than four times. However, the trade-off is lower performance. Figures 5 and 6 show the influence of UB_w on performance. The log block ratio is set to 5% in the BAST scheme. The result shows that the influence of UB_w on the performance is significant in NTFS. When UB_w is set to 4, the performance of the presented scheme is worse than that of BAST. For UB_w = 8 and above, our scheme delivers better performance than BAST. In RAND, the influence of UB_w is not significant. Thus, for high performance, we should set UB_w to a large value; 64 is reasonable because it is the same threshold as that of BAST. The secure file deletion tools can guarantee the complete removal of the sensitive contents by over-writing them at least 64 times.

5. Conclusion

In this work, we presented a method to implement secure file deletion for a page mapping scheme. Our scheme completely removes sensitive data by maintaining the physical locations of the invalidated data and the over-write count in the mapping table. If the over-write count exceeds a threshold, our scheme finds all the invalidated blocks corresponding to the logical page and removes them using a block erasure operation. A trace-driven simulation showed that the presented scheme delivers much better performance than the BAST scheme. The drawback is that greater memory is required to maintain the additional information. This memory overhead can be mitigated by loading a “hot” portion of the mapping table into memory and storing the rest in NAND flash memory. As future work, we plan to study the performance overhead of a scheme that loads only the hot portion of the mapping table.

Acknowledgements

This study was financially supported by Seoul National University of Science and Technology and by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2010-0003938).

References

[1] J. Kim, J. M. Kim, S. Noh, S. Min and Y. Cho, “A space-efficient flash translation layer for compactflash systems”, IEEE Transactions on Consumer Electronics, vol. 48, no. 2, **(2002)**.
[2] S. Lee, D. Park, T. Chung, W. Choi, D. Lee, S. Park and H. Song, “A log buffer based flash translation layer using fully associative sector translation”, ACM Transactions on Embedded Computing Systems, vol. 6, no. 3, **(2007)**.
[3] I. Shin, “Secure file delete in NAND-based storage”, International Journal of Security and Its Applications, vol. 6, no. 2, **(2012)**.
[4] P. Gutmann, “Secure deletion of data from magnetic and solid-state memory”, Proceedings of USENIX Security Symposium, **(1996)**.
[5] A. Kawaguchi, S. Nishioka and H. Motoda, “A flash-memory based file system”, Proceedings of USENIX Technical Conference, **(1995)**.
[6] A. Ban, Flash file system. U.S. Patent 5,404,485, **(1995)**.
[7] A. Ban, “Flash file system optimized for page-mode flash technologies”, U.S. Patent 5,937,425, **(1999)**.

Author

Ilhoon Shin received his B.S., the M.S., and the ph.D degrees in computer science and engineering from Seoul National University, Korea. He is currently an assistant professor of the department of electronics and information engineering at Seoul National University of Science & Technology. His research interests include storage systems, embedded systems, and operating systems.