Chen Liu*, Hoda Aghaei Khouzani, and Chengmo Yang

ErasuCrypto: A Light-weight Secure Data Deletion Scheme for Solid State Drives

Abstract: Securely deleting invalid data from secondary storage is critical to protect users' data privacy against unauthorized accesses. However, secure deletion is very costly for solid state drives (SSDs), which unlike hard disks do not support in-place update. When applied to SSDs, both *erasure-based* and *cryptography-based* secure deletion methods inevitably incur large amount of valid data migrations and/or block erasures, which not only introduce extra latency and energy consumption, but also harm SSD lifetime.

This paper proposes *ErasuCrypto*, a light-weight secure deletion framework with low block erasure and data migration overhead. *ErasuCrypto* integrates both erasure-based and encryption-based data deletion methods and flexibly selects the more cost-effective one to securely delete invalid data. We formulate a *deletion cost minimization* problem and give a greedy heuristic as the starting point. We further show that the problem can be reduced to a *maximum-edge biclique* finding problem, which can be effectively solved with existing heuristics. Experiments on real-world benchmarks show that *ErasuCrypto* can reduce the secure deletion cost of erasure-based scheme by 71% and the cost of cryptography-based scheme by 37%, while guaranteeing 100% security by deleting all the invalid data.

Keywords: SSD, secure deletion, cost optimization

DOI 10.1515/popets-2017-0009
Received 2016-05-31; revised 2016-09-01; accepted 2016-09-02.

1 Introduction

NAND flash memory-based Solid State Drives (SSDs), given their advantages of high storage density, low energy consumption, fast random access latency, and high shock resistance, are widely used in many computer systems as secondary storage. In the meantime, users expect SSDs to deliver the same features as traditional Hard Disk Drives (HDDs).

One important feature that concerns users' privacy is *secure deletion of invalid data* from the secondary storage, which should make *invalid data* irretrievable without affecting the accessibility of *valid data*. In traditional HDDs, many techniques [1–3] and tools [4–7] achieve secure deletion through directly overwriting the storage space of invalid data. This policy is not applicable to SSDs, however, as they adopt a more complicated data management scheme that brings extra challenges to secure deletion. Specifically, data are read from or write to the SSD at the granularity of a *page*. However, due to its physical attributes, flash memory only allows unidirectional write operations that change 1s to 0s. As a result, in SSDs, new data cannot be directly written to a physical page containing invalid data, but needs to be written to blank physical pages instead. This *out-of-place* update property makes all those overwriting-based secure deletion approaches [1–7] ineffective for SSDs. More crucially, this policy creates more invalid data in the SSD since invalid data are created not only upon deleting a file but also upon any update to the file.

To reuse invalid pages, the SSD controller has to trigger *erasure* processes that employ a unidirectional operation to change 0s to 1s. Leveraging this process to erase all the invalid data pages in the SSD is a straightforward methodology of secure deletion, which is referred to as *erasure-based* method in this paper. However, erasure can only be performed at the granularity of an entire *block*, which usually contains 64 to 256 pages. As a result, if a block selected for erasure contains valid pages, those pages must be migrated to other blocks in order to maintain their availability. Both erasure and the resultant valid-page migration are energy and time consuming. They are also harmful to SSD lifetime: each SSD block can only sustain 10^4 to 10^6 Write/Erase (W/E) cycles [8] before the block wears out and becomes unusablee.

An alternative method, which is referred to as *cryptography-based* in this paper, encrypts a data when writing it to the SSD. When the data becomes invalid, it can be deleted by erasing the corresponding key [9]. Although this method avoids the need for large amount

***Corresponding Author: Chen Liu:** University of Delaware, E-mail: liuchen@udel.edu
Hoda Aghaei Khouzani: University of Delaware, E-mail: hoda@udel.edu
Chengmo Yang: University of Delaware, E-mail: chengmo@udel.edu

of data erasures, since a unique key is shared by n pages, all the valid pages sharing the same key with a to-be-deleted invalid page have to be migrated to other places. On the other hand, when n=1, it is reported in [9] that the SSD throughput will be affected due to the large volume of key access requests.

To summarize, while both *erasure-based* and *cryptography-based* schemes guarantee deletion of all the invalid data, solely utilizing one of them will result in high erasure and/or data migration overhead. In this paper, we propose to integrate both methodologies into a framework named *ErasuCrypto*. While ensuring that all the invalid data are securely deleted, this framework brings the flexibility of selecting for each invalid page the cheaper way to delete it. In a nutshell, the contributions of this paper are the following:

- We identify the challenge of *secure data deletion* in SSDs, targeting *invalid data* created due to not only file deletion but also file updating.
- We analyze the cost of *erasure-based* and *cryptography-based* secure deletion methods, and propose a hybrid *ErasuCrypto* framework to delete invalid data while reducing the deletion cost.
- We propose and model the *deletion cost optimization* problem, and furthermore prove that the problem can be reduced to a *maximal-edge biclique* finding problem, which is proven to be NP-hard.
- We propose several heuristics to solve the *deletion cost optimization* problem.

The rest of the paper is organized as follows. Section 2 provides background knowledge on SSDs and reviews related works on secure data deletion. Section 3 introduces the threat model, the design goals, and the technical motivation. Section 4 proposes the secure deletion cost optimization problem and provides two heuristics. Section 5 describes the ErasuCrypto framework. Section 6 presents disk access traces based experimental results, while Section 7 concludes the paper.

2 Technical Background

2.1 Properties of Solid State Drives

SSDs typically adopt a four-level storage hierarchy. There are multiple *elements* in each flash memory while each element comprises multiple *planes*. Each plane includes multiple *blocks*, while each block consists of multiple *pages*, which are the smallest unit of SSD access. Fig. 1 shows a commodity 64GiB SSD with 8 elements;

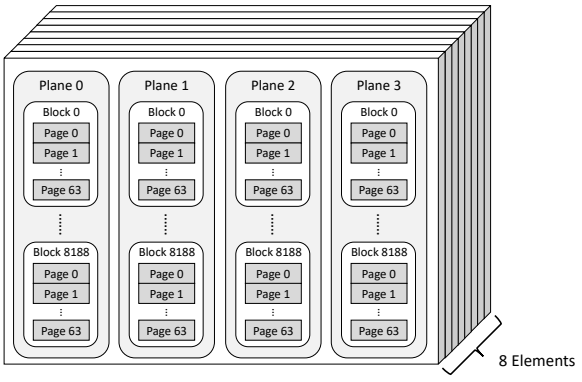

Fig. 1. The four-level hierarchy of SSD.

each element contains 4 planes, each comprising 2048 blocks of 64 4KiB pages.

One of the characteristics of flash-based storage is its out-of-place update property. There are three basic operations in SSDs: *read*, *write*, and *erase*. Write can only change a cell from 1 to 0, and a cell has to be erased to 1 before it can be re-written. *Read* and *write* are performed at the granularity of a *page*, while *erase* is performed at the granularity of a *block* (typically containing 64–256 pages). Thus intuitively, to update the data in a page, first all the valid pages in the block have to be stored in a buffer, and then after erasing the block, the buffered pages plus the newly updated page have to be written back to the block. In order to avoid this time- and energy-consuming process, SSDs do not directly update a written page, but write the new data to a free page. This property is known as *out-of-place update*. Due to this property, a physical page in an SSD has 3 states: *Free*, *Valid*, and *Invalid*. The erase operation changes pages from *Invalid* to *Free*, while the write operation changes pages from *Free* to *Valid*, and from *Valid* to *Invalid*.

An illustrative example is shown in Fig. 2. Fig. 2 (a) shows the state of pages before updating logical page LP_1 in the SSD. LP_1 is mapped to physical page 1 in block A. Block B that contains the pointer to the next free page to be written is called the *active block*. There is only one active block in each SSD plane. Fig. 2 (b) shows the effect of the update process: the old data in page 1 is marked as *invalid*, while LP_1 is re-mapped to page 5 in block B, which stores the new data.

Since the physical address of data changes upon each update, Flash Translation Layer (FTL) is added to the control layer of SSDs to keep track of the varying mapping between Logical Page Number (LPN) and Physical Page Number (PPN). Upon each write operation, the associated entry in the FTL is updated. FTL

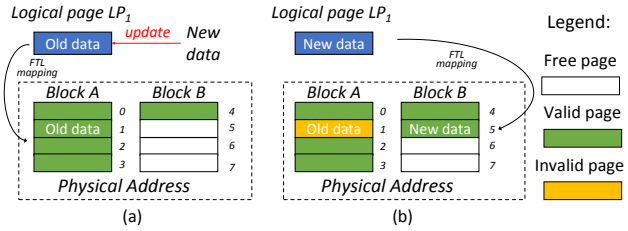

Fig. 2. Stale data generated by out-of-place update.

can be in page-level or block-level granularity, or even a combination of both [10, 11]. The coarser the FTL granularity, the less the storage overhead, but the worse the SSD performance and storage efficiency. Overall, many commercial SSDs adopt page-level FTLs [11].

As more writes are performed to the SSD, the number of *Invalid* pages increases. To recycle these *Invalid* pages, SSDs employ a Garbage Collector (GC), which selects blocks with a majority of *Invalid* pages, moves *Valid* pages in the selected blocks to other places, updates the FTL to record new page mappings, and erases the selected blocks to make them ready for future use. The frequency of GC invocation largely determines the Quality-of-Service (QoS) and the lifetime of SSDs. In most SSDs, GC is triggered only when necessary (i.e., when the number of free blocks fails below a threshold), and targets those young blocks (i.e., with below-average W/E counts) which have a majority of *Invalid* pages.

2.2 Related Work on SSD Secure Deletion

Although the out-of-place update property is profitable for reducing SSDs access latency, it induces additional privacy threats to SSDs. Specifically, invalid data will be generated not only upon file deletion, but also upon file update. This property can be clearly observed in Figure 2. When the logical page LP_1 is re-mapped to page 5 in block B, the old data in page 1 is not overwritten. Therefore, every "update" operation increases the number of invalid pages in the SSD. Secure deletion should target all the invalid pages in the SSD, regardless of whether they belong to a deleted file or not.

The secure deletion challenge in SSDs has already caught the attention of some research works. In [12], the authors propose a page-level deletion operation called *scrubbing*. The idea is to directly overwrite an invalid data page to turn all the bits in the page to 0, thus deleting the invalid data. In [13], a scheme combining garbage collection and per-page scrubbing is proposed for flash memory. In [14], scrubbing is also utilized to implement flash-based *physical unclonable functions* (PUF). However, due to *program disturb effect*[1] [15], scrubbing a page may introduce unpredictable errors to the pages within the same block. In other words, scrubbing may impact data accessibility of the valid data. This disturb effect is even more severe [15] in multi-level cell (MLC) flash memory, which is used more commonly in commercial products than single-level cell (SLC) flash. To minimize errors, MLC flash adopts a sequential programming strategy, and in-place reprogramming is neither supported nor suggested by flash manufacturers [15, 16].

Wei et al. [17] propose a "verifiable" SSD sanitation scheme called *Scramble and Finally Erase* (SAFE), with the goal of not only erasing all the data in an SSD, but also verifying that security erasure is successfully performed. They argue that encrypting data and deleting the key is fast but not verifiable, since reliably destroying the cryptography keys is challenging. On the other hand, erasing all the blocks in the SSD is slow but verifiable. The SAFE scheme combines both methods to provide a fast and verifiable secure deletion approach. In their model, all the data in the SSD is encrypted with a key. When performing SAFE, it first deletes the key to make the SSD "keyless", and then erases the entire SSD to make the deletion "verifiable". However, this technique can only sanitize the entire SSD. In other words, there will be no valid data left after the process.

Another set of research develops file system level secure deletion approaches for SSDs. In [18], a secure deletion approach for the YAFFS file system [19] is proposed. All the files are kept encrypted and the cryptography keys are stored at the header of each file. To securely delete a file, the block that contains the key will be erased. This method works well for deleting an entire file, but is not able to handle the invalid data generated due to file updates. *DNEFS* [9] is another file system level secure deletion scheme that leverages cryptography. It encrypts each *data node* (the unit of read/write in the file system) with a unique key. To delete a data node, the corresponding key will be deleted. However, as this scheme does not maintain a deterministic mapping between pages and key, it has to record, for each page, the position of its key in the header of the page.

A user-level secure deletion scheme named *purging* is proposed in [20], which requires no modifications to the storage system. Since the user does not have direct

1 The program operation on one page subjects the other pages in the same block a weak programming voltage. This phenomena is called *program disturb effect*. This effect is most significant to pages adjacent to the page being programmed [15].

control to the lower-level storage controller, a passive method is applied: the user should fill all the empty space in the file system with as much junk files as it can holds. This process will impassively trigger many garbage collection operations, thus securely deleting the invalid data. While this scheme ensures secure deletion, the purging processes incur high deletion latency and degrade SSD lifetime, especially when the empty space to be filled is large.

In addition to these research works, current SSDs also provide some built-in sanitation support. For example, both SCSI and ATA offer sanitation command [21] to erase all the data blocks in the SSD. However, sanitation cannot be used towards secure deletion, which should delete only invalid data while retaining valid data intact. Some commercial SSDs provide cryptography support for data accesses, such as Intel SSD 320 series and SSD 520 series [22]. Data stored in the SSD are in ciphertext form, while encryption/decryption is performed upon each SSD write/read access. In these SSDs, users' privacy is secured with a master password stored in BIOS. However, this scheme does not make invalid data unrecoverable. An adversary who steals the device can obtain the password stored in BIOS. Or, the user may be forced to surrender the password upon a subpoena. On the other hand, destroying the master key will sanitize the entire SSD, therefore is not preferable if the user still wants to access valid data.

3 Technical Motivation

3.1 Threat Model

We assume that the target computer system is equipped with an SSD as secondary storage. The attack method is the so-called *coercive attack* described in [9]. In particular, an attacker may utilize coercion method, such as legal subpoena, to force the user to disclose the device and any necessary information related to data privacy of the device. The attacker is then able to access any data, valid or invalid, stored on the device. With this threat model, encryption will be ineffective to protect data privacy since the key might be forfeited.

The user may request all the invalid data on the device, for example the photos or text messages previously deleted from the file system, become unrecoverable. However, in the meantime, all the valid data stored on the device should still be accessible. This goal is different from that of *disk sanitation*, which deletes all the data on the device, no matter whether they are valid or invalid.

The defenders in the threat model are SSD vendors. They will provide functionality to fulfill the secure deletion need of users, and flexibility to choose whether and when to trigger secure deletion. In addition, given the potentially high deletion cost, it is preferable to minimize the secure deletion cost. It is assumed that the SSD controller provides some cryptography support (e.g. AES [23, 24]) for data accesses, such as Intel self-encrypting SSDs [22], that the defender can take advantage of. The defender also has the flexibility to select different keys for different pages and determine where to store the keys.

One important criterion for evaluating defense schemes is the amount of information that needs to be kept in secret. It is noteworthy that our proposed scheme does not rely on any secret information. The cryptography keys are stored in the SSDs and can be obtained by the attacker if they are not erased. The attacker can extract logical-to-physical mapping information stored in the FTL, and may even know the exact defense scheme used in the SSD. In either case, the privacy level achievable by the proposed destruction scheme will not degrade.

3.2 Design Goals

Based on the threat model, a high-quality secure data deletion scheme should achieve the following two goals:

- **Securely delete all invalid pages**. After secure deletion, data in an invalid page must be irretrievable even if the page can be physically accessed. The constraint is that all the valid data should still be accessible after secure deletion, that is, they should either stay in their original locations without being touched, or be migrated to other pages. This goal evaluates the **effectiveness** of a scheme.
- **Minimize the secure deletion cost**. As mentioned before, the cost of secure deletion process can be measured by the number of block erasures and the number of page migrations. While the first goal must be satisfied, the deletion cost should also be minimized. This goal evaluates the **efficiency** of a scheme.

3.3 A Concrete Example

As briefly discussed in Section 1, the two secure deletion major methods, namely, the *erasure-based* and the *encryption-based* schemes, are able to meet the first design goal listed above. However, they both incur high deletion cost, as will be explained below.

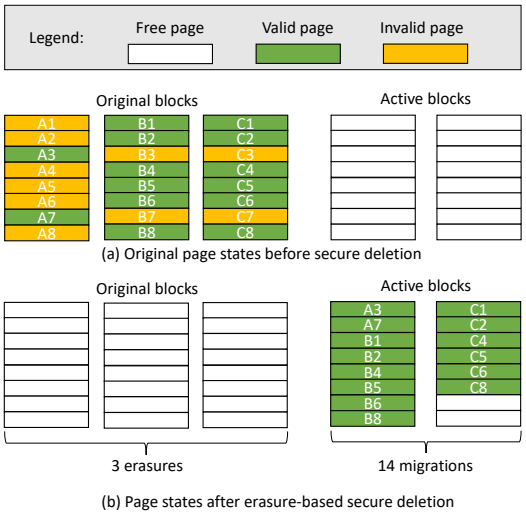

Fig. 3. Erasure-based secure deletion. The deletion cost includes 3 erasures and 14 migrations.

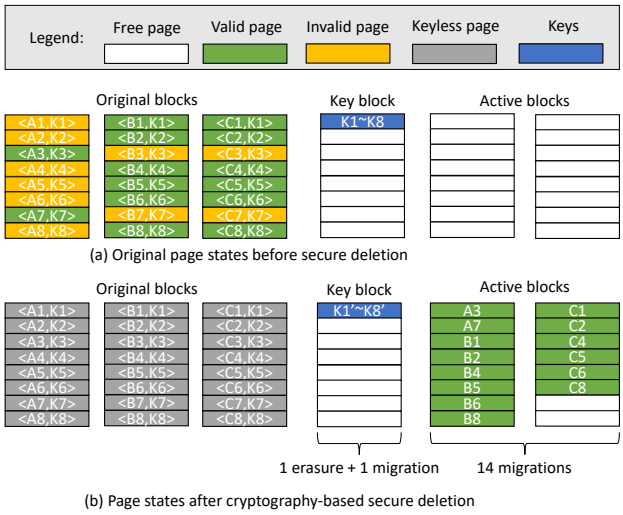

Fig. 4. Cryptography-based secure deletion. A data page with its key deleted is called *keyless*. The deletion cost includes 1 erasure and 15 migrations.

Fig. 3 provides a concrete example of the erasure-based scheme. There are three blocks in the example, each of which contains 8 pages. Fig. 3(a) shows the page states before initiating the deletion process. As can be seen, there are 10 invalid pages and 14 valid pages in total. Since the only way to delete an invalid page is to erase the block containing it, all the three blocks in Fig. 3(a), as they all contain invalid pages, have to be erased. As a result, all the valid pages in the three blocks have to be migrated to the other free pages before initiating the erasure operations. Fig. 3(b) shows the page states after applying the erasure-based secure deletion. As can be seen, all the valid pages were migrated to active blocks and the three original blocks were erased. The secure deletion cost of this process is **3** erasure operations plus **14** page migrations in total.

In comparison, Fig. 4 shows the same example but with a cryptography-based deletion scheme similar to the one proposed in [9]. In this example, all the pages are stored in encrypted form. As shown in Fig. 4(a), a total number of 8 keys ($K1$ through $K8$) are used and they are stored in a single page in the key block. One key is shared by 3 pages on the same row. For example, $A1$, $B1$, and $C1$ share the same key $K1$. Since the only way to securely delete an invalid page is to destroy the corresponding key, all 8 keys have to be destroyed. Hence, all the 14 valid pages need be migrated. Meanwhile, to destroy the keys, it is necessary to erase the key block and then write the new keys ($K1'$ through $K8'$) back to the key block, as shown in Fig. 4(b). This process does not require erasure to any data block, but still needs **1** erasure to the key block plus **1** migration of the key page and **14** migrations of the valid data pages.

As can be observed, under the constraint of deleting all the invalid pages in the SSD, these two approaches do not provide any flexibility in choosing a block to erase or choosing a key to delete. In the erasure-based approach, all the blocks that contain one or more invalid pages have to be erased. Similarly, in the cryptography-based approach, all the keys that are used to encrypt one or more invalid page have to be deleted. However, if these two approaches are both applied, it is theoretically possible to select a more cost-efficient way to securely delete an invalid page. Such flexibility is illustrated in Fig. 5, which applies different secure deletion methods to the invalid pages shown in Fig. 4(a). As can be seen, block A is erased since it contains a majority of invalid pages. On the other hand, the invalid pages in blocks B and C can be deleted by destroying keys $K3$ and $K7$. These operations together delete all the invalid pages, while the overhead is to migrate only two valid pages $A3$ and $A7$. All the other valid pages remain untouched. Taking the cost of key block erasure and migration into account, the total deletion cost in this example is **2** block erasures and **3** page migrations.

A side-by-side comparison of the costs of the three secure deletion approaches is listed in Table 1. As can be seen, the integrated method outperforms the erasure-based method by incurring both fewer erasures and fewer migrations. When compared to the cryptography-based method, the integrated method requires one more erasure but largely reduces the number of migrations

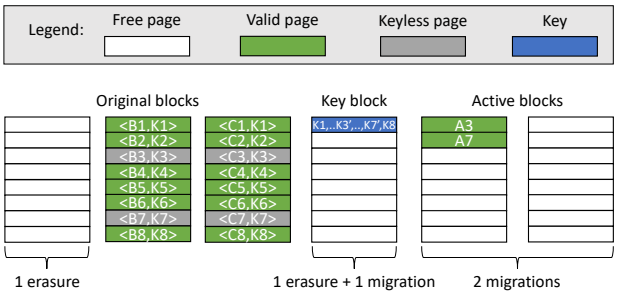

Fig. 5. Integrated secure deletion. The secure deletion cost includes 2 erasure and 3 migrations.

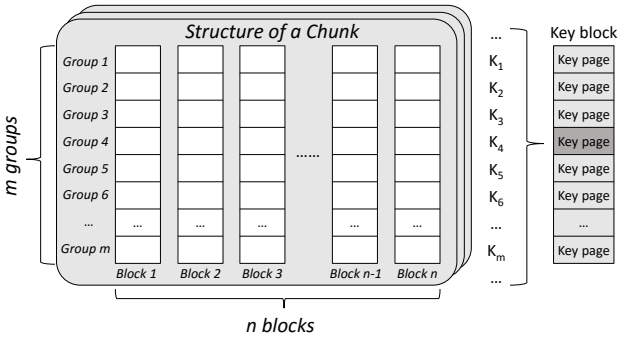

Fig. 6. Proposed SSD organization composed of chunks

Table 1. Secure deletion cost comparison

Method	Erasure	Migration
Erasure-based	**3**	**14**
Cryptography-based	**1**	**15**
Integrated	**2**	**3**

from 15 to 3. Clearly the benefit of low deletion cost is brought by the flexibility of the integrated method in selecting the more cost-effective deletion approach.

4 Deletion Cost Minimization Problem

In this section, the deletion cost minimization problem is defined, modeled, and solved. First of all, the storage structure to accommodate the proposed *ErasuCrypto* framework is provided. Then, the secure deletion problem is formalized as an integer linear programming (ILP) problem. A greedy heuristic is given as a straightforward solution to the problem. After that, we prove that the deletion cost minimization problem can be reduced to the *maximum-edge biclique* finding problem in a bipartite graph. Existing well-developed heuristics can be utilized to provide a nearly-optimal solution.

4.1 Proposed SSD Organization

Same as the crypto-based secure deletion methods [9], *ErasuCrypto* requires extra storage for the keys. Specifically, blocks in SSD are classified into two types: the majority are *data blocks* while the rest are *key blocks*, as shown in Fig. 6. The data blocks in each SSD element are partitioned into *chunks*. Assuming each chunk contains n data blocks[2] and each block contains m pages, a chunk can be represented as a $m \times n$ matrix. Each column in the matrix represents a block, and each row in the matrix is named as a *group*, which consists of the i^{th} pages of all the blocks in a chunk. Under this structure, a chunk can be considered as either "n blocks each containing m pages" or "m groups each containing n pages".

As shown in Fig. 6, all the pages in a group share one single encryption key, and a total of m different keys are required for each chunk. One benefit of such a structural organization is that there is no need to record the key location of any data block since it is always fixed and can be easily computed based on the corresponding group index. Furthermore, keys are stored densely in the dedicated *key blocks*. Assuming a page size of 4KiB and a key size of 128 bits, a total of 256 keys can be stored in a key page. If each block contains $m = 64$ pages, each key page can hold all the keys of up to 4 chunks, and each key block can hold all the keys of up to 256 chunks. This implies that the storage overhead of key blocks is only $1/256n$, which is completely negligible.

4.2 ILP Formulation of Deletion Cost Minimization

Since the chunks in the proposed SSD organization are non-overlapping, the secure deletion procedure of the entire SSD can be decomposed as a collection of the secure deletion procedures of all the chunks. In other words, invalid pages in the SSD can be deleted if and only if the invalid pages in each chunk are wiped out, while the overall deletion cost is minimized if and only if the deletion cost of each chunk is minimized. We define the *deletion cost minimization* problem of **a single chunk** as the problem of finding an optimal *secure deletion solution* for a chunk that satisfies the two design goals defined in Section 3.2.

2 The last chunk in an element may have $< n$ blocks if the total number of data blocks in an element is not divisible by n.

Here we formulate the deletion cost minimization problem in the language of integer linear programming (ILP). For each chunk, a two-dimensional *page state matrix* $State[m][n]$ is used to represent the states of all the pages in the chunk. For example, $State[i][j]$ $(1 \leq i \leq m, 1 \leq j \leq n)$ is the state of the i^{th} page in the j^{th} block. Each page has three possible states:

- $State[i][j] = 1 \iff$ the page is an *invalid page* that should be securely deleted.
- $State[i][j] = 0 \iff$ the page is a *valid page* that should still be accessible after secure deletion.
- $State[i][j] = -1 \iff$ the page is a *free page* that contains no data. No constraint is imposed on free pages.

A *secure deletion solution* consists of the selection of the block(s) to be erased and/or the key(s) to be deleted. To record the block erasure solution, a binary vector $Dblock[n]$ is used. $Dblock[j] = 1 (1 \leq j \leq n)$ if and only if block j is selected to be erased. Similarly, binary vector $Dgroup[m]$ is used to record the key deletion choices. $Dgroup[i] = 1 (1 \leq i \leq m)$ if and only if the key of group i is selected to be deleted. In the rest of the paper, a solution is called **feasible** if it satisfies the first design goal, that is, deleting all the invalid pages in the given state matrix, and **infeasible** otherwise.

The first design goal requests every invalid page being securely deleted, which can be modeled as follows:

- $\forall\, i, j$ $(1 \leq i \leq m, 1 \leq j \leq n)$:

$$Dgroup[i] + Dblock[j] \geq State[i][j] \tag{1}$$

In inequality (1), if $State[i][j]$ equals 1, at least one of $Dgroup[i]$ and $Dblock[j]$ should be 1. In other words, a feasible solution should select either the i^{th} row or the j^{th} column or both of them to delete the invalid page $State[i][j]$. On the other hand, if $State[i][j]$ is 0 or -1, which means the page is either valid or free, any values of $Dgroup[i]$ and $Dblock[j]$ will satisfy the inequality.

The second design goal is to minimize the secure deletion cost, which can be modeled as follows:

- $\forall\, i, j$ $(1 \leq i \leq m, 1 \leq j \leq n)$:

$$\#\ Mgr = \sum_{i=1}^{m} \sum_{j=1}^{n} (State[i][j] = 0 \wedge Dgroup[i] + Dblock[j] > 0) \tag{2}$$

$$\#\ Era = \sum_{j=1}^{n} Dblock[j] \tag{3}$$

$$\textbf{Goal: } minimize\ (\#\ Mgr + k \times \#\ Era) \tag{4}$$

In Equations (2)–(4), $\#\ Mgr$ stands for the number of valid pages to be migrated while $\#\ Era$ is the number of erasures to be performed. The goal is to minimize the overall detection cost, and k is an integer factor representing the ratio of the cost of one erasure over the cost of one migration. The factor can be configured by the user.

Note that this model simplifies the optimization goal by excluding the erasure and migration overhead of the key blocks. However, this will not affect the quality of the solution because such overhead is not only negligible compared to the costs incurred by the data blocks but almost invariant regardless of the secure deletion solution. As analyzed in Section 4.1, each key block can store the keys of up to 256 chunks. As long as any of these keys need to be deleted, the key block has to be erased and replaced with updated keys. In other words, due to the high key storage density, it is almost guaranteed that each key block will be erased and updated. On the other hand, the number of key blocks is only $1/256n$ of the number of data blocks when chunk size is n. Therefore, excluding such constant and negligible overhead from the optimization goal will not affect the quality of the solutions.

4.3 Greedy Heuristic

A straightforward greedy heuristic to minimize the deletion cost is shown in Algorithm 1. The idea is to iteratively select the block/group with the highest percentage of invalid pages, and erase/delete the block/key accordingly. During each iteration, a score is calculated for each group and block that contains invalid page, as shown in lines 5 and 8. Note that the erasure weight k is included in the denominator of B_{score} but not G_{score}, in order to prioritize groups over blocks when they have the same percentage of invalid pages. After selecting a group/block with the highest score, all the counters will be updated accordingly in lines 13 to 21. Then the states of all the pages in the selected group/block will be set to -1 (line 22), to indicate the row/column being "deleted". The procedure terminates only when all the invalid pages are covered, as shown in line 3. This guarantees that the obtained solution is feasible.

Regarding the complexity of the greedy algorithm, the initialization part requires scanning the entire $State[i][j]$ matrix and hence has a complexity of $O(mn)$. During each iteration of the *while* loop, scores of all the groups and blocks are calculated, resulting a complexity of $O(m+n)$. When updating the counters, each page in the block/group will be checked, which has a complexity of $O(\max(m, n))$. Since each iteration guarantees the deletion of either a row or a column, there will be at

Algorithm 1: Greedy heuristic

input : $State[m][n]$ – Chunk state matrix
$B_{invalid}[n]$ – Block invalid-page counts
$B_{valid}[n]$ – Block valid-page counts
$G_{invalid}[m]$ – Group invalid-page counts
$G_{valid}[m]$ – Group valid-page counts
$Invalid$ – Total invalid-page counts
k – Erasure weight

output: A feasible solution given by $Dblock[n]$ and $Dgroup[m]$

```
1  begin
2      Scan State[m][n] and initialize B_invalid,
         B_valid, G_invalid, G_valid, and Invalid;
3      while Invalid>0 do
4          for i ← 1 to m has G_invalid[i] > 0 do
5              G_score[i] ← G_invalid[i] / (G_invalid[i] + G_valid[i])
6          end
7          for i ← 1 to n has B_invalid[i] > 0 do
8              B_score[i] ← B_invalid[i] / (B_invalid[i] + B_valid[i] + k)
9          end
10         Select the group/block S with the highest
             G_score/B_score to delete;
11         Set the S entry in Dblock[n]/Dgroup[m]
             to 1;
12         for every State[i][j] in group/page S do
13             if State[i][j] = 1 then
14                 Invalid − −;
15                 G_invalid[i] − −;
16                 B_invalid[j] − −;
17             end
18             if State[i][j] = 0 then
19                 G_valid[i] − −;
20                 B_valid[j] − −;
21             end
22             State[i][j] ← −1;
23         end
24     end
25 end
```

most $m+n$ iterations. Therefore, the while loop has a complexity of $O((m+n)^2)$. Overall, the complexity of obtaining a feasible deletion solution for a given chunk is $O((m+n)^2)$. Since m is a constant for a given SSD while the number of chunks in a SSD is proportional to $1/n$, the overall complexity of obtaining a feasible deletion solution for the entire SSD is $O((m+n)^2/n) = O(n)$. This indicates that the complexity of the greedy algorithm is linearly proportional to n.

4.4 Graph-based Heuristic

While the greedy heuristic ensures a feasible solution, it does not effectively bound the secure deletion cost. In this part, we will present a better solution by proving that the *deletion cost minimization* problem can be reduced to a *maximum-edge biclique finding* problem and hence can be solved with well-designed heuristics.

4.4.1 Maximum Edge Biclique Problem Formulation

Lemma 4.1. *Minimization goal (4) is equivalent to the following maximization goal (7):*

– $\forall\, i, j\ (1 \le i \le m, 1 \le j \le n)$:

$$\#\ \overline{Mgr} = \sum_{i=1}^{m}\sum_{j=1}^{n}(State_{i,j} = 0 \wedge Dgroup_i + Dblock_j = 0) \tag{5}$$

$$\#\ \overline{Era} = \sum_{j=1}^{n}(1 - Dblock_j) \tag{6}$$

$$\textbf{\textit{Goal:}}\ maximize\ \#\ \overline{Mgr} + k \times \#\ \overline{Era} \tag{7}$$

Proof. $\overline{Mgr}$ in Definition (5) is the number of valid pages that do not need to be migrated during secure deletion, while $\overline{Era}$ in Definition (6) is the number of blocks that do not need to be erased. Therefore we have:

$$\begin{aligned} &Goal\ (4) + Goal\ (7) \\ &= (\#Mgr + k \times \#Era) + (\#\overline{Mgr} + k \times \#\overline{Era}) \\ &= (\#\overline{Mgr} + \#Mgr) + k \times (\#\overline{Era} + \#Era) \\ &= \sum_{i=1}^{m}\sum_{j=1}^{n}(State_{i,j} = 0) + k \times n \end{aligned} \tag{8}$$

As can be seen, the sum of $\#\overline{Mgr}$ and $\#Mgr$ is the total number of valid pages, while the sum of $\#\overline{Era}$ and $\#Era$ is n. Since both sums are constant, the sum of Goal (4) and (7) is also constant. Therefore, minimizing Goal (4) is equivalent to maximizing Goal (7). □

Assuming that the solution is feasible and there is no free page in the chunk (the impact of free pages will be discussed later), a page which is not covered by any selected row or column is guaranteed to be valid. As a result, the term $State[i][j] = 0$ can be omitted from the expression of $\overline{Mgr}$. The maximization goal (7) can be rephrased as maximizing the product of two factors: the

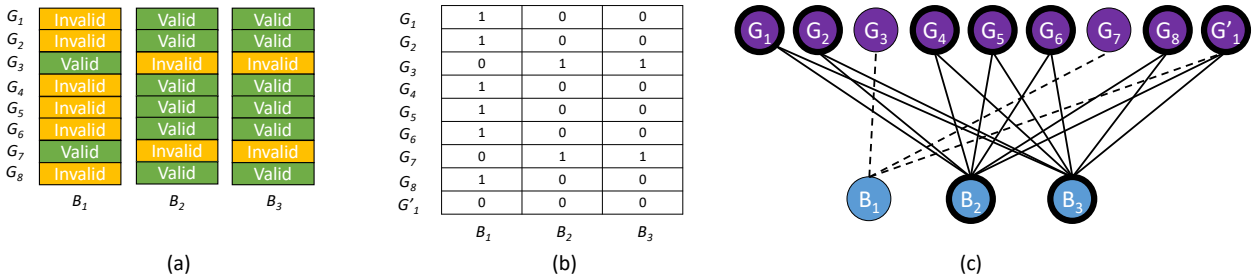

Fig. 7. (a) The original page state matrix. (b) The augmented page state matrix when $k = 1$. (c) The augmented page state bipartite graph. Edges correspond to 0s in the augmented page state matrix. Vertices and edges that form the maximum-edge biclique are respectively marked with bold outlines and solid lines.

unselected groups plus a constant k, and the unselected blocks, as shown in the following steps:

$$
\begin{aligned}
&Goal\ (7)\\
&= \#\ \overline{Mgr}\ +\ k \times \#\ \overline{Era}\\
&= \sum_{i=1}^{m}\sum_{j=1}^{n}(Dgroup[i] = 0\ \wedge\ Dblock[j] = 0)\\
&+ k \times \sum_{j=1}^{n}(1 - Dblock[j])\\
&= \sum_{i=1}^{m}(Dgroup[i] = 0) \times \sum_{j=1}^{n}(Dblock[j] = 0)\\
&+ k \times \sum_{j=1}^{n}(Dblock[j] = 0)\\
&= (\sum_{i=1}^{m}(Dgroup[i] = 0) + k) \times \sum_{j=1}^{n}(Dblock[j] = 0)
\end{aligned}
\tag{9}
$$

Equation (9) motivates us to transform the matrix to a bipartite graph[3] and find its maximum-edge biclique[4]. Specifically, for each chunk, an *augmented page state matrix* can be constructed by appending k rows with all valid pages, called *shadow groups*, to the original state matrix $State[m][n]$. An example is shown in Fig. 7. The original state matrix shown in Fig. 7 (a) is the same as the example used in Section 3.3. In this case, $k = 1$ and an extra group G'_1 is appended to the original matrix to build the augmented page state matrix, as presented in Fig. 7 (b).

Based on the augmented page state matrix, a bipartite graph named *augmented page state bipartite graph* can be built. The two sets of vertices respectively represent all the blocks (columns) and all the groups (rows) in the matrix, including the shadow groups. As shown in Fig. 7 (c), there is an edge E_{ij} between group node i and block node j if and only if $State[i][j]$ in the augmented page state matrix is valid. The maximum-edge biclique of this bipartite graph is represented by bold circles and solid lines in Fig. 7 (c).

With the bipartite graph representation, the problem of finding the optimal deletion solution can be reduced to the problem of finding the maximum-edge biclique. This will be proved by showing the feasibility and the efficiency of the solution.

Theorem 4.2. *Given any biclique in the augmented page state bipartite graph, a feasible secure deletion solution is to erase/delete all the blocks/groups that are* ***not*** *included in biclique.*

Proof. If the solution is infeasible, there must be an invalid page $P[i][j]$ left undeleted. Since the solution erases/deletes all the blocks/groups not included in the biclique, both the block vertex $B[i]$ and group vertex $G[j]$ of P should be included in the biclique. However, this is impossible since a biclique is a complete bipartite graph and there must be an edge connecting $B[i]$ and $G[j]$. As mentioned before, an edge in the augmented page state bipartite graph indicates that the corresponding page P is valid, which contradicts the assumption. Therefore, the assumption of the solution being infeasible is false. □

Theorem 4.3. *Assume there is no free page in the chunk. Given a feasible solution, the* ***not-erased*** *blocks and the* ***not-deleted*** *groups constitute a biclique in the augmented page state bipartite graph.*

3 A bipartite graph is a graph whose vertices can be divided into two disjointed sets U and V, and each edge in the graph connects a vertex in U to a vertex in V.

4 A biclique is a complete bipartite graph with an edge connecting any vertex in U to any vertex in V. A maximum-edge biclique of a bipartite graph is its subgraph which is a biclique and has the maximum number of edges.

Proof. If the graph containing the *not-erased* blocks and the *not-deleted* groups is not a biclique, there must be two vertices $B[i]$ and $G[j]$ in the graph that are not connected. If so, the corresponding page $P[i][j]$ must be invalid and has not been securely deleted, which contradicts the given assumption. □

Theorem 4.2 proves the *sufficient* condition that any biclique in the augmented page state bipartite graph derives a feasible solution to the secure deletion problem, while Theorem 4.3 proves the *necessary* condition that a feasible solution is always derived from a biclique in the graph. Therefore, when there is no free page in the chunk, the optimal secure deletion solution is guaranteed to be among all the solutions derived from the bicliques. In the next step, we will show that the optimal solution can be derived from the *maximum-edge biclique* of the augmented page state bipartite graph.

Lemma 4.4. *A maximum-edge biclique in the augmented page state bipartite graph must include all the shadow group vertices.*

Proof. Assume a shadow group vertex G'_i is not included in the maximum-edge biclique. Since there is an edge between any block vertex and G'_i, adding G'_i to the current maximum-edge biclique will generate a bigger biclique with more edges, which contradicts the definition of maximum-edge biclique. Therefore the maximum-edge biclique must include all the shadow group vertices. □

Theorem 4.5. *Given the maximum-edge biclique of an augmented page state bipartite graph, erasing/deleting all the blocks/groups that are* ***not*** *in the biclique guarantees a feasible and optimal solution that minimizes the secure deletion cost.*

Proof. On the block vertex side of the biclique, the total number of vertices is $\sum_{j=1}^{n}(Dblock[j] = 0)$ since all the not-erased blocks are included in the biclique. On the group vertex side of the biclique, the total number of vertices is $\sum_{i=1}^{m}(Dgroup[i] = 0) + k$ since all the not-deleted groups are included and all the shadow groups are included according to Lemma 4.4. Furthermore, since a biclique is a complete bipartite graph, the total number of edges in it is the product of the numbers of the two sets of vertices, which is $(\sum_{i=1}^{m}(Dgroup[i] = 0)+k) \times \sum_{j=1}^{n}(Dblock[j] = 0)$. This is the same as Equation (9), the rephrased representation of Goal (7). According to Lemma 4.1, maximizing Goal (7) is equivalent to minimizing Goal (4). Therefore, the optimal secure deletion solution can be derived from the maximum-edge biclique of the augmented page state bipartite graph. □

4.4.2 Biclique Finding Heuristic

According to Theorem 4.5, a feasible and min-cost secure deletion solution can be obtained by finding the maximum-edge biclique in the corresponding augmented page state bipartite graph. According to [25], finding the maximum-edge biclique in a given bipartite graph is an NP problem. Therefore, we utilize the heuristic proposed in [26] to get near-optimal solutions. The work approximates the maximum-edge biclique problem as a *rank-one non-negative factorization* problem named $R1N_d(G)$. It also proves that finding the stationary points of $R1N_d(G)$ can localize biclique in the bipartite graph. Therefore, nonlinear optimization can be used to find good stationary points of $R1N_d(G)$, which correspond to bicliques with large number of edges. Based on this observation, a biclique finding heuristic with the complexity of $O(E)$ (E is the number of edges in the bipartite graph) is proposed in [26]. When applied to the proposed secure deletion problem, the sum of all the edges in all the bipartite graphs is approximately the total number of valid pages in the SSD. Therefore, the overall complexity is a constant independent of the group size n, implying that this heuristic is less complex than the greedy heuristic in Section 4.3.

4.4.3 Impact of Free Pages

The proofs shown in Section 4.4.1 assume the absence of free pages in a chunk. In this part, we analyze the impact of free pages. Fundamentally, blocks in the SSD can be categorized into the following three types based on the existence of free pages in them:

- **No free page**. Most of in-use blocks have no free page but only valid and/or invalid pages in them.
- **Free page only**. All the free blocks in the SSD have only free pages.
- **Mixture**. The block under written, called active block, has a mixture of free pages and valid/invalid pages. Due to the sequential programming property of Flash memory, there is one and only one active block in each SSD plane.

Since the first case has already been analyzed before, here we only discuss the second and the third cases. First, a block containing only free pages can be completely omitted from secure deletion since it on one hand has no invalid data to delete, and on the other hand has

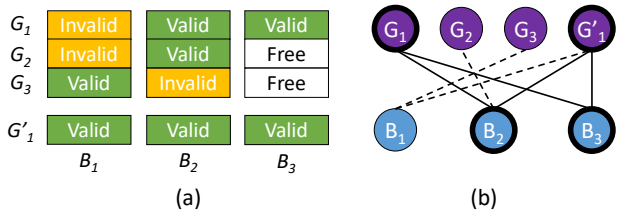

Fig. 8. (a) An augmented page state matrix with free pages. (b) The maximum-edge biclique is marked with solid lines. However, the solution is not optimal since deleting key G_2 is unnecessary.

no valid page to be migrated. These blocks can be excluded from the corresponding page state matrix. The matrix will have $< n$ columns, but the quality of the solution will not be affected.

On the other hand, when a chunk contains a block of the third category, the solution obtained from the maximum-edge biclique may not deliver the minimum secure deletion cost. A concrete example is shown in Fig. 8. Fig. 8 (a) shows a 3×3 augmented page state matrix wherein one shadow group is appended (i.e., $k = 1$). The corresponding bipartite graph is shown in Fig. 8 (b), wherein the vertices and edges of the maximum-edge biclique are marked with solid lines. According to Theorem 4.5, the min-cost deletion solution is to erase B_1 and delete keys of G_2 and G_3 since they are not included in the biclique. However, deleting G_2 is unnecessary since the invalid page located at row 2 column 1 is already handled by erasing B_1. In fact, the min-cost deletion solution is to erase B_1 and delete G_3.

The fundamental reason for this non-optimality is that when a block contains free pages, Theorem 4.3 is not longer true. If there is no edge connecting block vertex $B[i]$ and group vertex $G[j]$ in the augmented page state bipartite graph, the corresponding page $P[i][j]$ may not be invalid but be free instead. Therefore it is not guaranteed that the optimal solution is among the ones derived from bicliques.

However, the impact of such non-optimality is negligible given the existence of at most one mixture block in each SSD plane. As a result, in each place there will be at most one chunk that contains a mixture block who may affect the optimality of its security deletion solution. For a standard configuration of a plane with 8192 blocks and a group size of 8, optimal solutions can still be obtained for over 99.9% of the chunks. On the other hand, for the only exceptional chunk with the mixture block, feasibility of the solution obtained with maximum-edge biclique finding is still guaranteed since Theorem 4.2 still holds even with the existence of mixture blocks.

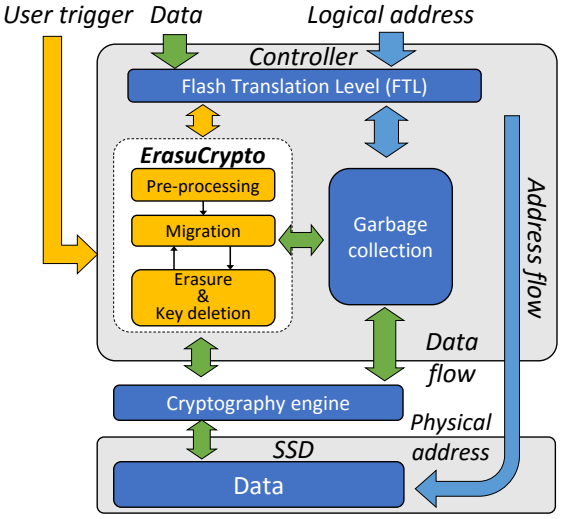

Fig. 9. SSD controller enhanced with secure data deletion

5 ErasuCrypto Framework

Figure 9 presents an overview of the various components in the proposed ErasuCrypto framework. The SSD controller is extended to incorporate ErasuCrypto. This implies that the secure data deletion process is transparent to the user. However, an interface is provided for the user to manually trigger a secure deletion process on demand. Furthermore, the deletion process consists of the following three steps:

- **Pre-processing**. In the first step of secure deletion, the FTL is accessed to obtain page state information. Then the heuristics described in Section 4 are used to analyze each chunk and select the block(s) to erase and the group key(s) to delete.
- **Valid data migration**. This step ensures the accessibility of the valid data. All the valid pages located in the selected blocks or groups are migrated. These pages are first encrypted with their new keys, and then written to the active block in the corresponding SSD plane. The FTL is updated to include new page mapping information.
- **Erasure & Key deletion**. In this step, the deletion engine interacts with the garbage collection engine and the SSD to erase the selected blocks and delete the selected keys.

As secure deletion is performed at the granularity of chunks, theoretically the second and third steps can be performed to all the chunks simultaneously since the operations to different chunks are independent. However, in real-world situation the maximum chunk processing throughput is bounded by two factors. One is the I/O

capacity of the SSD which typically allows one chunk to be accessed at a time for each SSD element. The second factor is the free space available in each SSD element, which should be large enough to hold all the valid pages that need to be migrated. Based on these observations, the chunks in each element are processed one by one so as to make maximum utilization of the SSD access capacity while relaxing the stress on free space. In this way, blocks erased as a result of the already processed chunks can be used to fulfill migration needs of the to-be-processed chunks. Furthermore, when the total number of free blocks drops below the threshold, garbage collection is invoked to recycle the blocks with a majority of invalid keyless pages.

The secure deletion processes of different chunks can be further pipelined to improve the throughput. To be more specific, the pre-processing step is done by the co-processor in the SSD controller and requests no SSD access. Therefore this step can be done in parallel with the second and third steps that access the SSD. For example, when chunk A is at the second or the third step, the next chunk B can be pre-processed simultaneously.

Another design detail is the handling of key blocks. The deletion of keys is an update process consisting of first erasing the key block and then writing the new keys back to the block. Since a key block holds keys of many chunks, it is preferable to contiguously process all these chunks and buffer their keys in the DRAM buffer of the SSD controller, so as to accelerate key accesses during the deletion process and avoid repetitive key block updates. After processing all the chunks mapping to the same key block, the key block is first erased and then updated with the new keys. Note that the key blocks will not become the bottleneck of SSD lifetime since they are written and erased only once during each deletion process, while data blocks are frequently written and erased to serve regular SSD accesses.

6 Experimental Results

6.1 Methodology

We have extended the Microsoft SSD simulator *SSD-Model* [27] to implement the proposed secure data deletion method as well as the previous work. SSDModel is built upon DiskSim 4.0 [28], an event-driven simulator. The SSD under the test is 64GiB in size and contains 8 NAND-Flash elements. Each element has a dedicated event queue, thus accesses to different elements are handled independently. Each element consists of 4 planes, each plane contains 8192 blocks, and each block is composed of 64 pages whose size is 4KiB.

Table 2. SSD access latency and energy consumption [29, 30]

	Latency(ms)	Energy(μJ)
Page migration	0.225	42.76
Block erasure	1.500	527.68

Table 3. Benchmark information

	Function	Duration	Total Requests
hm	Hardware monitoring	54hr	2,828,089
proj	Project directories	85hr	4,943,216
rsrch	Research projects	119hr	3,242,437
stg	Web staging	76hr	4,377,032

Regarding the chunk size, it affects both the migration cost and the SSD read/write response time. While a smaller group size reduces the migration costs of the proposed work and the cryptography-based secure deletion, it increases the number of keys to read from the SSD for read/write operations, thus degrading I/O throughput. The study in [9] shows that when group size is set to one, the read and write throughputs are decreased by 23% and 19%, respectively. To balance regular SSD performance and migration cost during secure deletion, we use a group size of 8 in our experiments.

The parameter k in the optimization goal (4) is set to 7 to model the fact that the block erasure latency is about 6.67 times of the migration latency, as shown in Table 2. We select four real-world traces from the Microsoft I/O trace sets [31]. Their characteristics are summarized in Table 3. All of these traces are collected from real-world working servers. Table 3 shows that these traces display a relatively heavy server workload at about 14 requests per second, while the size of each request is one page. The heavy workload not only makes the performance of the storage system critical, but also generates a large number of invalid data. For instance the benchmark *stg*, which is a web staging server, may have many temporary files and cookies which, from user's perspective, are already deleted. The benchmark *hm*, which is a hardware monitoring server, may contain many system failure reports which, from system administrator's perspective, are also deleted. To protect user's privacy, those invalid data should be permanently removed with secure deletion approaches.

In the simulator, a secure deletion request is implemented as an event inserted to the event queue of each SSD element. It is served by the SSD controller when reaching the head of the queue. This allows us to extract

the latency of the deletion procedure. Our studies investigate two different secure deletion triggering conditions:

- **At-the-end**: This is the user triggering secure deletion at the end of each trace. When the trace reaches to the point 5-hour before the finish point, a deletion request is placed in the event queue of each SSD element.
- **In-the-middle**: This is the user triggering secure deletion in the middle of program execution. Without loss of generality, a deletion request is placed in the event queue at the middle point of each trace.

The results of five secure deletion schemes are compared in the section, including:

- The **Erasure-based** scheme which erases all the blocks containing invalid pages.
- The **Cryptography-based** scheme which deletes all the keys used by one or more invalid pages.
- The **ErasuCrypto-Greedy** scheme which uses the greedy heuristic in Section 4.3 to select group keys to delete and blocks to erase.
- The **ErasuCrypto-Graph** scheme which uses the graph-based heuristic in Section 4.4 to select group keys and blocks.
- The **ErasuCrypto-ILP** scheme which uses ILP to solve the deletion cost minimization problem optimally. The results of this scheme serve as the lower bounds of secure deletion costs.

Neither the **Erasure-based** nor the **Cryptography-based** scheme is based on any specific previous work. Instead, they are implemented following the methods introduced and discussed in Section 3.3. Another noteworthy point is that the overhead of key block erasure and key migration is included when counting the numbers of erasures and migrations.

6.2 Simulation Results

Since all the secure deletion schemes guarantee the deletion of all the invalid data, only the secure deletion cost and hence its latency and energy impacts will be presented and compared.

6.2.1 Secure Deletion Costs

The first set of experiments compares the secure deletion costs of different schemes under the two deletion triggering conditions listed before. Theoretically, the erasure-based scheme should have the highest deletion cost since it needs to erase all the blocks that contain invalid pages and migrate all the valid pages in those blocks. On the other hand, the cryptography-based scheme only requires the erasure of the key blocks, but needs to migrate all the selected keys and all the valid data encrypted with the selected keys. As a result, it should display low erasure counts and large migration counts. The three ErasuCrypto schemes are expected to incur deletion costs lower than both the the erasure-based and the cryptography-based schemes, while ErasuCrypto-Graph is expected to outperform ErasuCrypto-Greedy.

Fig. 10 (a)–(c) show the results of different schemes when triggering secure deletion at the end of each trace. Fig. 10 (a) presents the overall secure deletion cost of the SSD, calculated by summing up the deletion costs of all the chunks. These results are consistent with the theoretical analysis above. The erasure-based and the cryptography-based schemes always have the highest and the second highest deletion costs, respectively. Both ErasuCrypto heuristics are able to reduce the deletion costs close to the lower bounds given by ILP. This confirms that both the greedy and the graph-based heuristics are very effective in generating near-optimal solutions. What is more, the graph-based heuristic consistently outperforms the greedy heuristic. In fact, it has the exact same cost as the optimal, since the distribution of invalid data in real-world traces is not random enough to let the graph-based heuristic reach to a non-optimal point.

Since the secure deletion cost is jointly determined by the erasure count and the migration cost, Figs. 10 (b) and 10 (c) respectively report these two values of the different deletion schemes. Note that the erase and migration operations performed by garbage collection due to lack of free space are not counted here but counted in the next set of results. The erasure-based scheme engenders a larger number of erasures ($> 30k$) than all the other schemes, which inevitably degrade SSD lifetime. The cryptography-based scheme only engenders a few erasures (< 150) which are all performed to the key blocks. The ErasuCrypto schemes have more erasures than the cryptography-based scheme, but the values are still order of magnitude less than those of the erasure-based scheme.

In terms of the number of valid page migrations, Fig. 10 (c) shows that both the erasure-based and the cryptography-based schemes have significant migration counts. The cryptography-based scheme outperforms the erasure-based scheme in all the benchmarks except for *rsrch*. Such variation is related to the distribution of valid and invalid pages within each chunk. If the distribution of invalid pages in the columns is more biased than that in the rows, the migration counts of

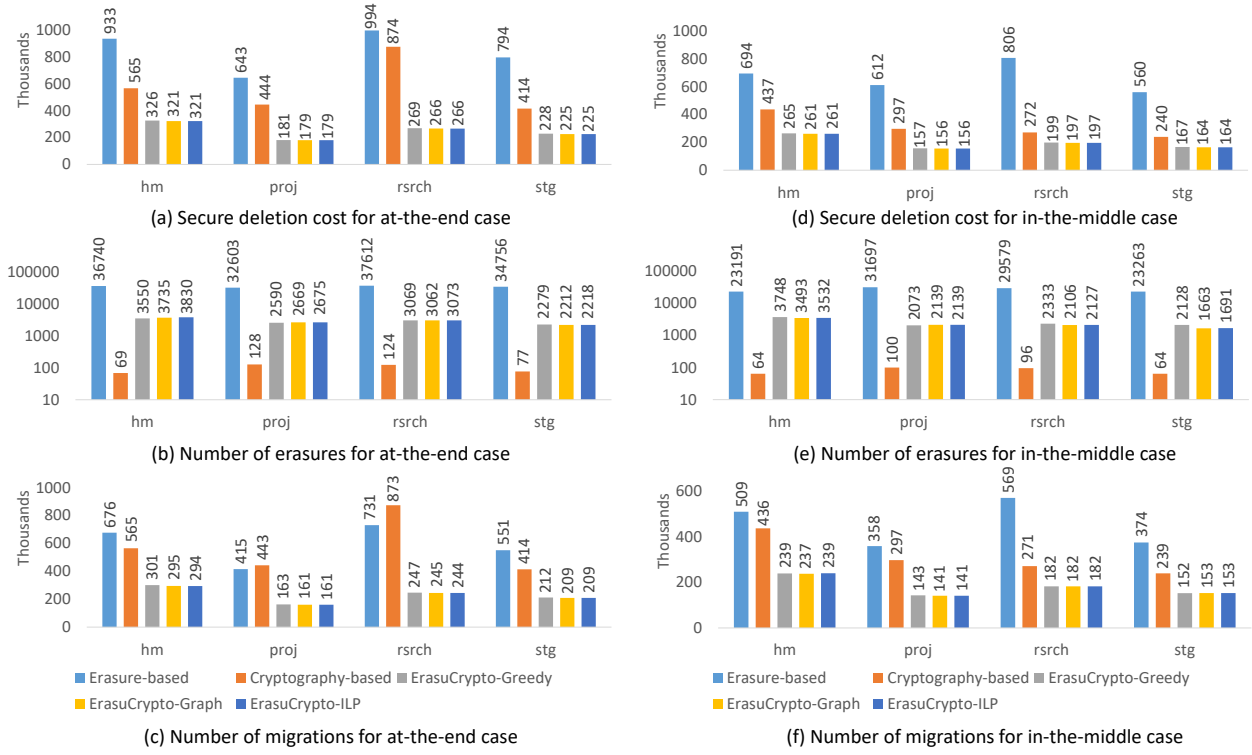

Fig. 10. Comparison of different secure deletion schemes in terms of deletion cost, number of erasures, and number of migrations.

the cryptography-based scheme would be higher, and vice versa. Fig. 10 (c) also shows that the ErasuCrypto schemes consistently outperform both the erasure-based and the cryptography-based schemes across all the benchmarks. As expected, ErasuCrypto-Graph is closer to the lower bound than ErasuCrypto-Greedy.

Fig. 10 (d)–(f) respectively show the overall deletion cost, erasure count, and migration count when performing secure deletion in the middle of each trace. The trends in these graphs are consistent with those in Fig. 10 (a)–(c). One interesting observation is that for all the benchmarks, the deletion costs are lower than those of the at-the-end deletion case. This is probably because the traces have fewer invalid pages to delete when the execution is half-way through. While the exact deletion cost may vary under different triggering conditions, ErasuCrypto is always able to outperform the erasure-based and the cryptography-based schemes and incur the lowest deletion costs.

6.2.2 Impact on Regular Garbage Collection

In addition to the deletion cost evaluated above, a secure deletion process may have non-trivial impact on regular program execution. To evaluate such impact, we also extract the migration and erasure counts of the **baseline**, which executes the traces without any secure deletion. In the baseline, erasures are triggered only when the percentage of free blocks is lower than a threshold (for example $< 5\%$) in the SSD. The garbage collector selects blocks with a majority of invalid pages to erase, and migrates valid pages in those blocks, if any. Since the secure deletion process may vary the percentage of free blocks, it will inevitably influence the number of garbage collections and the number of migrations during garbage collection. Fig. 11 reports those impacts by presenting the number of erasures and migrations performed during garbage collection. Again, both the *at-the-end* and *in-the-middle* triggering conditions have been studied.

Figs. 11 (a) and 11 (c) show the erasure counts of the baseline and the secure deletion schemes. Under either deletion triggering condition, the erasure-based scheme consistently has smaller erasure counts than the baseline. This is because the secure deletion process engenders a large number of erasures, as shown in Figs. 10 (b) and 10 (e), thus increasing the percentage of free blocks in the SSD. Therefore, garbage collections are rarely triggered after the deletion process. In contrast, the cryptography-based scheme leads to much larger erasure counts than the baseline since it triggers very few erasures during secure deletion but needs to migrate a lot of valid data to free blocks. As a result, the

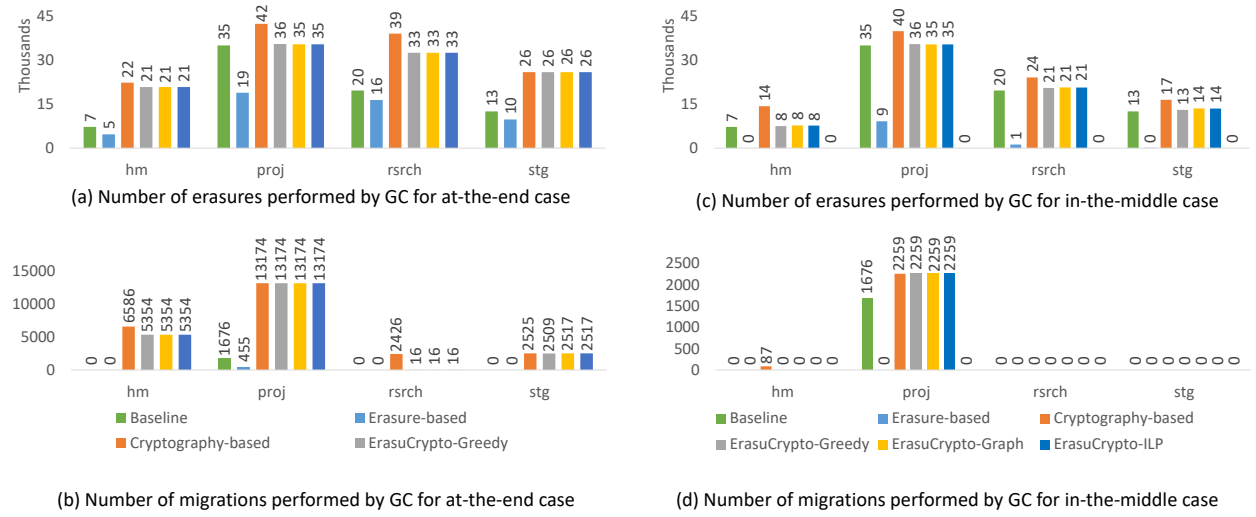

Fig. 11. Impact of secure deletion on garbage collection in terms of number of erasures and migrations performed by garbage collector.

SSD has fewer free blocks, and garbage collections will be triggered more frequently after and even during the deletion process. Regarding the ErasuCrypto schemes, since they integrate both the erasure-based and the cryptography-based schemes, their erasure counts are higher than those of erasure-based and the baseline, but lower than those of cryptography-based.

Figs. 11 (b) and 11 (d) present the migration counts of different schemes. The baseline garbage collection incurs no migration for all the benchmarks except for *proj*. This confirms the effectiveness of garbage collection in selecting blocks with a majority of invalid pages to erase and recycle. The erasure-based scheme also has very few migrations during garbage collection since it already largely reduces the number of garbage collections. The cryptography-based scheme, on the other hand, has the largest amount of migrations. A detailed study shows that these migrations are incurred during the deletion process, when garbage collection is invoked to recycle blocks to hold those migrated valid data. The ErasuCrypto schemes have migration counts slightly lower than the cryptography-based scheme since they face lower pressure on the number of free blocks.

Comparing the two deletion triggering conditions, it can be observed that the in-the-middle condition engenders fewer block erasures and page migrations than the at-the-end condition, consistently for all the secure deletion schemes. This difference is due to two reasons. First, at the middle point of the trace, there are fewer invalid pages to delete, and the deletion process incurs fewer migrations and consumes fewer free blocks. In fact, all deletion schemes incur no or negligible amount of migrations for benchmarks *hm*, *rsrch*, and *stg*. Second, the erase operations performed during secure deletion create more free blocks, which can be used to serve SSD write requests during the second half of the trace. In sum, performing the secure deletion in the middle of the trace is more cost-effective, while performing deletion at the end delivers higher privacy level. Overall, it is still the user's choice to determine when to trigger a secure deletion process.

6.2.3 Latency and Energy Consumption

Fig. 12 compares the latencies and energy consumptions of the secure deletion schemes, under the *at-the-end* and *in-the-middle* triggering conditions, respectively. The latency and energy values presented in Table 2 are used, while the latency of the entire deletion process is the time period between inserting the secure deletion requests to the event queues and finishing serving the deletion request.

As can be seen, the erasure-based scheme always displays the highest latency and energy consumption, while the cryptography-based scheme has the second highest values. These data are consistent with the erasure and migration counts shown in Fig. 10. The two schemes perform the largest numbers of erasures and migrations, thus incurring high latency and energy overhead. The ErasuCrypto schemes have similar latency and energy consumption values, which are much lower than the erasure-based and cryptography-based schemes. The differences between the three ErasuCrypto schemes are too small to observe in the figure. Detailed examination shows that the graph-based heuristic is closer to optimal than the greedy heuristic.

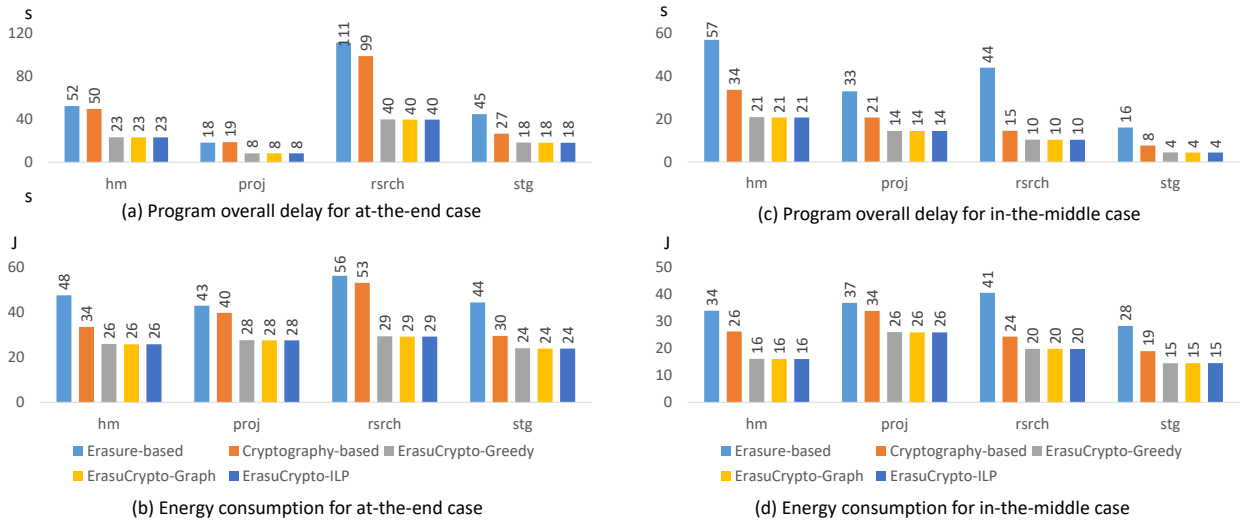

Fig. 12. Overall delay (in seconds) and energy consumption (in joules) of secure deletion process.

7 Conclusion

This paper has presented *ErasuCrypto*, a low-overhead and lifetime-friendly secure deletion framework for SSDs. It enhances user's privacy by securely deleting invalid data with two different deletion methods: erasing the blocks containing invalid data, or destroying the keys used to encrypt invalid data. The papers shows that exclusively applying either method incurs high deletion cost, but the integration of the two methods brings more flexibility which can be exploited to largely reduce the deletion cost. The paper has mathematically modeled the secure deletion cost minimization problem and proven that this problem can be reduced to the problem of finding *maximum-edge biclique* in a bipartite graph. It has also presented two heuristics to solve the problem: a greedy heuristic and a graph-based heuristic. The simulation results show that ErasuCrypto is able to reduce the secure deletion cost of erasure-based scheme by 71% and the cost of cryptography-based scheme by 37%, while still guaranteeing the deletion of all the invalid data.

References

[1] U. S. Defense Security Services, "Office of the Designated Approving Authority (ODAA) Process Manual," Nov. 2013. [Online]. Available: http://www.dss.mil/documents/odaa/ODAA%20Process%20Manual%20Version%203.2.pdf

[2] Royal Canadian Mounted Police, "G2-003, Hard Drive Secure Information Removal and Destruction Guidelines," Oct. 2003.

[3] S. L. Garfinkel and A. Shelat, "Remembrance of Data Passed: A Study of Disk Sanitization Practices," *IEEE Security & Privacy*, vol. 1, no. 1, pp. 17–27, Feb. 2003.

[4] M. Gauthier and D. Jagdmann, "srm - Linux man page." [Online]. Available: https://www.mankier.com/1/srm

[5] B. Durak, "wipe - Linux man page." [Online]. Available: http://manpages.ubuntu.com/manpages/wily/man1/wipe.1.html

[6] "Eraser." [Online]. Available: http://eraser.heidi.ie/

[7] "Disk Wipe." [Online]. Available: http://www.diskwipe.org/

[8] ITRS, "International Technology Roadmap for Semiconductors," Emerging Research Devices (ERD). [Online]. Available: http://www.itrs2.net/2013-itrs.html

[9] J. Reardon, S. Capkun, and D. A. Basin, "Data Node Encrypted File System: Efficient Secure Deletion for Flash Memory," in *Proceedings of the 21th USENIX Security Symposium*, Aug. 2012, pp. 333–348.

[10] T. Wang, D. Liu, Y. Wang, and Z. Shao, "FTL^2: A Hybrid Flash Translation Layer with Logging for Write Reduction in Flash memory," in *Proceedings of the 14th ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems*, May 2013, pp. 91–100.

[11] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings," in *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2009, pp. 229–240.

[12] M. Wei, L. M. Grupp, F. E. Spada, and S. Swanson, "Reliably Erasing Data from Flash-Based Solid State Drives," in *9th USENIX Conference on File and Storage Technologies*, Feb. 2011, pp. 105–117.

[13] K. Sun, J. Choi, D. Lee, and S. Noh, "Models and Design of an Adaptive Hybrid Scheme for Secure Deletion of Data in Consumer Electronics," *IEEE Transactions on Consumer Electronics*, vol. 54, no. 1, pp. 100–104, Feb. 2008.

[14] Y. Wang, W.-K. Yu, S. Wu, G. Malysa, G. E. Suh, and E. C. Kan, "Flash Memory for Ubiquitous Hardware Security Functions: True Random Number Generation and Device

Fingerprints," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, May 2012, pp. 33–47.

[15] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf, "Characterizing flash memory: Anomalies, observations, and applications," in *42nd Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2009, pp. 24–33.

[16] J. Reardon, D. A. Basin, and S. Capkun, "SoK: Secure Data Deletion," in *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, May 2013, pp. 301–315.

[17] M. Wei and S. Swanson, "SAFE: Fast, Verifiable Sanitization for SSDs," Unviersity of California, San Diego Computer Science & Engineering, Tech. Rep., Oct. 2010. [Online]. Available: https://cseweb.ucsd.edu/~swanson/papers/TR-cs2011-0963-Safe.pdf

[18] J. Lee, S. Yi, J. Heo, H. Park, S. Y. Shin, and Y. Cho, "An efficient secure deletion scheme for flash file systems," *Journal of Information Science and Engineering*, vol. 26, no. 1, pp. 27–38, Jan. 2010.

[19] C. Manning, "How YAFFS Works," 2010. [Online]. Available: http://dubeyko.com/development/FileSystems/YAFFS/HowYaffsWorks.pdf

[20] J. Reardon, C. Marforio, S. Capkun, and D. A. Basin, "User-level secure deletion on log-structured file systems," in *7th ACM Symposium on Information, Compuer and Communications Security*, May 2012, pp. 63–64.

[21] G. F. Hughes, T. Coughlin, and D. M. Commins, "Disposal of disk and tape data by secure sanitization," *IEEE Security & Privacy*, vol. 7, no. 4, pp. 29–34, Aug. 2009.

[22] Z. Balshai, O. Barel, D. DeVetter, E. Kaufman, and O. Livne, "Managing Intel Solid-State Drives Using Intel vPro Technology." [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/it-management-wde-ssd-amt-encryption-paper.pdf

[23] NIST, "Federal Information Processing Standard 197, The Advanced Encryption Standard (AES)." [Online]. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[24] M. Solanki, S. Salehi, and A. Esmailpour, "LTE Security: Encryption Algorithm Enhancements," in *2013 ASEE Northeast Section Conference*, Mar. 2013.

[25] R. Peeters, "The Maximum Edge Biclique Problem is NP-complete," *Discrete Applied Mathematics*, vol. 131, no. 3, pp. 651–654, Sep. 2003.

[26] N. Gillis and F. Glineur, "A continuous characterization of the maximum-edge biclique problem," *Journal of Global Optimization*, vol. 58, no. 3, pp. 439–464, Mar. 2014.

[27] Microsoft, "SSD Extension for DiskSim Simulation Environment," 2008. [Online]. Available: http://research.microsoft.com/en-us/downloads/

[28] J. S. Bucy, J. Schindler, S. W. Schlosser, and G. R. Ganger, "The DiskSim Simulation Environment," 2008. [Online]. Available: http://www.pdl.cmu.edu/DiskSim/

[29] S. Park, Y. Kim, B. Urgaonkar, J. Lee, and E. Seo, "A comprehensive study of energy efficiency and performance of flash-based SSD," *Journal of Systems Architecture: the EUROMICRO Journal*, vol. 57, pp. 354–365, Apr. 2011.

[30] P. Huang, K. Zhou, and C. Wu, "ShiftFlash: Make flash-based storage more resilient and robust," *Performance Evaluation*, vol. 68, Nov. 2011.

[31] D. Narayanan, A. Donnelly, and A. Rowstron, "Write Off-Loading: Practical Power Management for Enterprise Storage," *ACM Transactions on Storage*, vol. 4, no. 3, pp. 10:1–10:23, Nov. 2008.