Black-box forensic and antiforensic characteristics of solid-state drives

Gabriele Bonetti · Marco Viglione · Alessandro Frossi · Federico Maggi · Stefano Zanero

Received: 13 January 2014 / Accepted: 24 July 2014 / Published online: 9 August 2014

Abstract Solid-state drives (SSDs) are inherently different from traditional drives, as they incorporate data-optimization mechanisms to overcome their limitations (such as a limited number of program-erase cycles, or the need to blank a block before writing). The most common optimizations are wear leveling, trimming, compression, and garbage collection, which operate transparently to the host OS and, in certain cases, even when the disks are disconnected from a computer (but still powered up). In simple words, SSD controllers are designed to hide these internals completely, rendering them inaccessible if not through direct acquisition of the memory cells. These optimizations may have a significant impact on the forensic analysis of SSDs. The main cause is that memory cells could be preemptively blanked, whereas a traditional drive sector would need to be explicitly rewritten to physically wipe off the data. Unfortunately, the existing literature on this subject is sparse and the conclusions are seemingly contradictory. In this work we propose a generic, practical, test-driven methodology that guides researchers and forensics analysts through a series of steps that assess the "forensic friendliness" of a SSD. Given a drive of the same brand and model of the one under analysis, our methodology produces a decision tree that can for instance help an analyst to determine whether or not an expensive direct acquisition of the memory cells is worth the effort, because optimizations may have rendered the data unreadable or useless. Conversely, it can be used to assess the antiforensic techniques that stem from the characteristics of a given hardware, and to develop novel ones that are specifically suited to particular drives. We apply our methodology to three SSDs produced by top vendors (Samsung, Corsair, and Crucial), and provide a detailed description of how each step should be conducted. As a result, we provide two use cases, a test-driven triage classification of drives according to forensic friendliness, and the development of an anti-forensic technique specifically suited to a given drive.

G. Bonetti · M. Viglione · A. Frossi · F. Maggi · S. Zanero (✉)
DEIB, Politecnico di Milano, Milan, Italy
e-mail: stefano.zanero@polimi.it

G. Bonetti
e-mail: gabriele.bonetti@mail.polimi.it

M. Viglione
e-mail: marco.viglione@mail.polimi.it

A. Frossi
e-mail: alessandro.frossi@polimi.it

F. Maggi
e-mail: federico.maggi@polimi.it

1 Introduction

Solid-state drives (SSDs) have reached remarkable popularity nowadays, as their increasing capacity and affordable prices made them a good alternative to traditional, platter-based hard drives (HDD, from hereinafter) [12]. SSDs offer the flexibility and compatibility of HDDs, along with the shock-resistance ensured by the lack of mechanical components typical of flash drives, and the speed offered by flash memories.

SSD have a shorter lifespan than HDDs. NAND-based flash chips, in fact, have a physical limit of approximately 10,000 program-erase cycles. When approaching this limit, NAND floating gates exhibit problems in retaining their charge and, if not constantly refreshed, they lose their content. This means that keeping an SSD without power for several days may lead to data loss, which is obviously unac-

ceptable. While 10,000 cycles may seem a very high number, it is a rather low lifespan for hard drives. Another limita-tion is that whenever a block needs to be rewritten, it must be blanked first, causing a non negligible overhead in SSD drives, which use 16–512 kB blocks.

SSD vendors have developed specific techniques (such as write caching, trimming, garbage collection and compres-sion) to reduce the actual number of physical program-erase cycles. With these optimizations, SSD controllers make the already thick layers (e.g., operating system, driver, firmware) that over the years became necessary to bridge the application level with the physical level, even thicker. Indeed, previous work [11] demonstrated that, when it comes to implement-ing sensitive operations such as secure deletion, the entire data storage path needs to be taken into account, with unsus-pectable complexity.

As a consequence of this complexity, existing and widely adopted forensic data acquisition and analysis procedures may not be completely suitable for SSDs (e.g., the hash of an SSD may not be "stable" over time, as obsolete data may be automatically wiped by internal optimizations). In some cases, the only viable option is a white-box acquisition that bypasses the controller and reads the content of the NAND chips. Unfortunately, as explained in Sect. 2.1, a white-box acquisition is expensive, not always feasible, can possibly disrupt the drive, and may lead to the conclusion that data is lost or damaged. In this regard, it would be useful to have a simple and affordable (black-box) triage procedure to decide whether a white-box analysis may produce some usable outcome on given the SSD brand, model and release, and OS.

Looking at the problem from a dual perspective, a skilled attacker could develop an anti-forensic approach specifi-cally suited to that combination in order to avoid detection. Therefore, forensic analysts and security researchers may be interested in exploring and exploiting the peculiarities of a given device to obtain anti-forensic effects that are specifi-cally suited to that combination of SSD model, firmware and OS.

In this work we propose a generalized, practical analysis methodology to selectively address the peculiarities of SSDs that may impact forensic acquisition and reconstruction. Our methodology is a test-driven workflow that guides the foren-sic analyst through a series of experiments. The goal of each experiment is to assess how the controller logic behaves under different conditions and provide the analyst with useful insights on how the SSD under examination works and what are the optimizations adopted. Given the SSD model, brand and release, and the OS (if any) used on that SSD, our work-flow provides (i) insights on the potential impacts of such optimizations on the results of standard forensic tools, (ii) a practical decision framework to determine the expected suc-cess rate of retrieving lost data through white-box analysis, and (iii) useful information to build, or detect, anti-forensic approaches that are unique to that specific hardware/software combination.

As our methodology is black-box, it is transparently applicable to any SSD brand and model. Throughout this paper, we show this by applying our workflow on three SSDs of different vendors, each with a different controller, chosen because they are the most used ones: we cover this way a very vast variety of devices on the market and can analyze their peculiar behaviors, directly tied to the controller they are built with. Regardless of the specific experiments that we carry out for the sole purpose of demonstrating the practical-ity of our workflow, we show that a forensic analyst can use our tests to assess whether a certain feature is implemented in an arbitrary SSD.

Our original contributions can be summarized as follows:

- We propose a test-driven, black-box methodology to determine whether a SSD implements trimming, garbage collection, compression and wear leveling.
- We show our methodology by applying it on three popular SSD brands and models, and detail precisely how each step is conducted and how the results of each step are interpreted.
- We show how our the outcome of our methodology guides the practitioners in understanding how they impact their chances of data retrieval using traditional black-box, or expensive white-box analysis techniques.
- We show how the findings can be practically used to develop an anti-forensic technique that makes use of the specific peculiarities of one of the devices, and how this can be used by a forensic expert to try to avoid falling victim of one such technique.

The remainder of this paper is structured as follows. In Sect. 2 we describe the relevant state of the art that we analyzed to derive the research gaps defined in Sect. 3. In Sect. 4 we overview our methodology, which is detailed in Sect. 5. In Sect. 6 we provide two use cases that show the type of conclusions that can be drawn by applying our methodol-ogy. In Sect. 7 we critically discuss our methodology and propose the directions for future follow-up work in this area.

2 Background and related work

SSDs employ a complex architecture, with many hardware and software layers between the physical memory packs and the external interface to the computer. These layers, merged in the flash translation layer (*FTL*)[14], are in charge of reading and writing data on the ATA channel on one side and on the memory chips on the other side, as well as to com-

press, encrypt or move data blocks to perform optimizations. In HDDs the OS has direct access to the data contained on platters, and the controller is mostly limited to moving the magnetic head and read or write data. Instead, the FTL of SSDs performs much more complex functions: It translates logical block addresses (LBA) as requested by the OS into the respective physical block addresses (PBA) on memory chips. The underlying mapping is completely transparent and can be modified by the FTL at any time for any reason. The need for mechanisms such as the FTL has been studied extensively in [19], who analyze the lifespan of cells and their likeli-hood of losing their ability to retain data. They show that the endurance of memories greatly varies among the vendors and chip models, and that premature decay is caused by stressing cells with continuous writes.

2.1 White-box forensics analysis

The action of the FTL is transparent to software and to the host OS: To the best of current knowledge, there is no way to bypass the FTL via software, and explore the raw content of the memory chips. Such access requires tampering with the hardware, in what is commonly called a *white-box* acquisition.

Although a complete white-box analysis of a SSD is theoretically possible and in some cases feasible, it is also very difficult, time consuming and expensive, because no sin-gle hardware tool or methodology can help with every SSD drive, since each of them has a different architecture, different chips positioning, and many other details that make "generic" hardware tools impossible to build. Creating custom hardware requires the forensics analyst to acquire specific skills, buy expensive equipment, and, once a successful acquisition is finally carried out, spend a significant amount of time to reverse the implementation of the policies of SSD controllers.

In fact, [8] showed that it is possible to acquire data from a flash memory chip in several ways. One option is to use flasher tools that interact with the chip directly via the pins of the board; other ways are the use of a JTAG port usually left by vendors on devices to bypass the controller or, in extreme cases, the physical extraction of the chip for dumping via standard readers.

We performed an exploratory experiment with the three drives mentioned in the following (Samsung, Corsair, and Crucial) and limited hardware resources, and found it very hard even to access the chips on the board, without disrupting them, or to find accessible JTAG ports: Understandably, vendors tend to protect their intellectual property (i.e., the FTL algorithms) by discouraging this kind of access to the hardware.

To examine the feasibility of white-box acquisition, we attempted to read directly from the memory chips using non-expensive clips[1] but we obtained a fragmented, incomplete raw file. In addition, SSDs use different flash memory chips, with very different working parameters and low-level requirements. In some cases, it may be physically impossible to connect to the memory chips because of the way they are soldered to the board; as a consequence the chip often needs to be removed, thus potentially damaging the drive or destroying the evidence.

Similarlyto[8,18] also addressed data acquisition from flash memory chips, but at a lower level. His technique, however, is not suitable for forensic purposes because of the non-optimal recovery rate. This is actually a general concern with white-box methodologies: SSD controllers are specifically built to reach high throughput by leveraging parallel reads and writes; custom forensic hardware is much slower and can read only one chip at a time, making the dump (and the reconstruction) of an entire drive a very long process.

The state of the art in white-box analysis is the work by [9] and [20], who built a complete custom setup to interact with flash memory chips using an FPGA and several custom wing boards to enhance its compatibility. Although their goal is to enable easy development and prototyping of FTL algorithms, compression, cryptography and sanitization protocols, the same setup can be theoretically used to re-implement part of the FTL functionalities to ease white-box acquisition of SSDs. However, the internals of a controller are definitely proprietary information, unlikely to be documented publicly or even privately shared.

Indeed, the analyst would need to know any compression or obfuscation algorithm in use, as well as the data allocation policy (i.e., how bytes are spread over the memory chips). Otherwise it would be close to impossible to reconstruct files directly from the acquired data, as traditional file carvers are likely to fail. Although [6] showed how is possible to analyze a raw flash dump and reconstruct files without prior knowledge of the disposition of blocks performed by the FTL, their technique works only with small capacity chips (in the order of hundreds of megabytes).

[16] concentrated on FAT structure and demonstrated how to rebuild audio and video files from dumped NAND memories; their work suffers from the very same shortcoming: It is tailored for small amounts of data (e.g., cellphone memories). Also, data reconstruction is made even more difficult by SSD controllers because they often make use of data parallelism over the flash memory chips on the board.

2.2 Black-box forensics analysis

Differently from white-box approaches, *black-box* approaches read data as presented to the ATA interface by the SSD controller. Such approaches are by far more convenient and

[1] We used the TSOP NAND clip socket, available online for 29USD.

practical than white-box ones, and, most importantly, guarantee that the SSD is not damaged in the process. However, as we observed in Sect. 1, treating SSDs just like HDDs with black-box tools ensures only partial observability over the controller's behavior.

A seminal paper is [5], where the authors analyzed the file recovery rate on SSDs versus HDDs during a standard, black-box forensic acquisition. When issued a quick format command, the SSD used in the experiment wiped the entire content irrecoverably in a matter of minutes. They confirmed this result with a write blocker (i.e., re-attaching the SSD to a write blocker after the quick format), showing that this deletion did not happen as a result of commands issued by the host or its OS: SSDs can indeed schedule and perform their own write operations. This work provided one of the first hypothesis on how garbage-collection algorithms work, stating that some of them (primarily Samsung) are capable of "looking at the used/unused aspects of an NTFS filesystem by examining the free space bitmap". The authors hypothesized that these controllers may be file-system aware, and need no OS intervention to blank unused blocks. This would pose major issues, rendering traditional forensic methodologies (such as the use of write blockers) insufficient to preserve the digital evidence. However, as we report in Sect. 5.2, we were unable to replicate their experiment, even using the same OS, scripts, drive (including firmwares and versions) and working conditions. Even with the authors' help, we were unable to find the reason for this difference.

Similarly to [5,15] performed experiments on 16 different SSDs: They simulated real usage scenarios and tested the block-level recoverability. Each scenario was replicated under three OSs (Windows XP, Windows 7 and Ubuntu Linux 9.04). Their conclusion is that different combinations of usage, OS and file size influence the forensic recoverability of the SSD. Although this is by far the most exhaustive test on SSDs before our own, the authors focus solely on data deletion as effect of TRIM and garbage collection, without generalizing their findings. What is missing is an in-depth study of the correlation between the environment conditions (e.g., OS, filesystem, file size), the internal state of a disk (e.g., amount of free space, wear) and the corresponding behavior of the SSD. Our work goes beyond [15] because we designed and evaluated a comprehensive, test-driven methodology to fully understand the reasons behind each specific behavior.

[4] analyzed the behavior of one of the first SSDs with respect to file deletion. He wrote some typical files such as documents and images on an NTFS-formatted SSD, and then erased them to see how much data was recoverable afterward. Surprisingly, none of the files was recoverable via carving. These experiments, however, were narrowed to a single scenario and did not take into account all the possible factors that our methodology accounts for. For instance, as detailed in Sects. 4.2 and 5, thanks to our methodology we can explain why caching is the most reasonable explanation of [4]'s odd results. Last, [4] focused on TRIM, whereas our methodology considers the features implemented in most of the SSDs currently on the market.

3 Research challenges and goal

Having examined the relevant state of the art, we notice a relevant research gap that has practical implications. When applied to SSDs, black-box and white-box approaches have symmetric advantages and drawbacks: the former fails when a drive performs internal optimizations silently, whereas the latter fails on proprietary hardware, which is difficult to manipulate and access, or when data is compressed or encrypted. Black-box methods are faster and cheaper, white-box methods, if feasible, can yield better results. Our key observation is that a black-box triage is a mandatory prerequisite before committing resources to a challenging, costly and potentially fruitless white-box analysis.

With the above rationale in mind, we conclude that as every drive and every controller behave differently one from the other, it is important to provide a general methodology to perform forensically sound tests and determine how the FTL of a given SSD affects the results that standard forensic techniques may yield. To this end, we propose an analysis methodology that advances the state of the art for its generality, and hopefully offers a useful reference for forensic practitioners and researchers. Moreover, we provide pure black-box techniques to "estimate" the likelihood of retrieving additional data through a white-box effort, allowing forensic experts to triage evidence and avoid wasting resources. Last, we strive to replicate and validate the experiments described in the literature to take into account the previous conclusions in our methodology.

4 Methodology overview

The input of our methodology is an SSD of the same brand and model of the one under examination. The first step is to conduct a series of tests that determines whether the SSD implements certain features (regardless of the features stated by the vendor, which often turn out to be incorrect). If they are implemented, in some cases we are able to determine how fast and aggressive they are, and therefore how much they would influence data reconstruction.

Our methodology covers the following aspects:

TRIM This functionality mitigates the limitation that requires any block to be blanked before it can be rewritten. The trimming function erases data blocks that have been marked

as "deleted" by the OS. Trimming has an obvious negative impact on forensic analysis, as on-disk data persistence after deletion is no longer guaranteed: Once a block is marked as free by the OS, the controller decides when to blank it according to its policies. As noted in [5], this can occur regardless of data connection between the SSD and a host computer (e.g., during an acquisition, even when write blockers are used). Our methodology can determine the percentage of blocks that get erased and how fast this happens (Sect. 5.1).

Garbage collection (GC) This is a functionality that SSD vendors often tout as capable of greatly improving performance. However, as explained in Sects. 4.1 and 5.2, it is an ill defined concept. [5] hypothesizes that GC works by making the controller somehow filesystem-aware, and thus able to infer on its own which blocks are obsolete by monitoring the file-allocation table. If this were the case, GC would bias forensic acquisitions significantly. GC is not triggered by the OS; consequently, data could be erased whenever the disk is powered on, even if no write commands are issued, thus rendering write blockers and other precautions useless. Therefore, it is important to analyze this feature. Our methodology can determine whether a form of GC is active.

Erasing patterns Some SSDs show peculiar behaviors when using TRIM: They do not erase all the marked blocks but rather a subset of them based on some drive's internals. Our methodology explores this behavior to characterize the erasing patterns (Sect. 5.3).

Compression Some drives transparently compress data to use less blocks and reduce cell wearing. Compression poses no direct challenges in black-box forensics acquisitions, whereas it makes white-box analysis challenging, since the data read directly from the chips would be unusable, unless the compression algorithm were known or reverse engineered. Therefore, in our methodology we included a step that can verify whether compression is active.

Wear leveling (WL) This functionality reduces the usage of flash cells by spreading their consumption as evenly as possible across the drive. The easiest and least expensive wear-leveling implementations activate if certain blocks have reached an excessive number of writes compared to the rest of the disk, in which case the FTL writes the new block on a less-used portion of the disk, and updates the internal file mapping table that the FTL maintains [2]. Alternatively, ven-dors sometime provide the disk with extra physical space, so that new blocks can be written on brand new memory cells. This second technique is used, for example, by the Corsair F60 SSD, which has a total of 64 GB flash memory but allows to address only 60 GB at a time; the drive itself is more expensive for the vendor, but if the wear leveling functional-ity is correctly implemented it grants a much longer lifespan. Since wear leveling is standard in modern drives, it is useful to explore whether the SSD implementation masks the effect of the so-called "write amplification" (see Sect. 5.5).

Files recoverability Even though blocks may not be erased, they might be changed or partially moved, making it impossible to retrieve them through carving. This test checks how many deleted files can be retrieved from the drive (Sect. 5.6).

We combine the analysis of all these factors in a ranking of a drive in terms of its "forensic friendliness", as detailed in Sect. 6.1.

Our methodology covers all known combinations of factors that may trigger each feature. In addition, our methodology is designed to avoid redundant tests, which would certainly not trigger any of the features. When feasible, we compare our results with the outcome of previous studies. In particular, we validate the experiments of [5] on garbage col-lection, which is a particularly controversial and ill-defined topic as detailed in Sect. 4.1.

4.1 Garbage collector vs. garbage collection

The difference between garbage collector and garbage collection is a controversial concept that needs to be clari-fied before explaining our methodology. Many works in the literature treat these two features as being substantially the same and propose methods to trigger and reverse-engineer this functionality. They are, however, two different (logical) components of an SSD controller, which unfortunately share a very similar name causing considerable confusion.

For the purpose of our work, the garbage collect*or* is the *process* that deals with unused blocks—as a garbage collector does with unused variables in modern memory managers. The internals of SSD garbage collectors are not disclosed, and they may vary from drive to drive. However, it is known that the garbage collector is tightly tied to the TRIM functionality: among other capabilities, it has access to the "preemptive-erase table" (Sect. 5.1) filled by TRIM and takes care of physically wiping trimmed blocks. Additionally, the garbage collector helps wear leveling by moving around blocks whenever the wear factor of a cell is beyond a certain threshold.

On the other hand, vendors never disclosed any details about the garbage collect*ion functionality*, although there were some speculations about how it is supposed to work. The only known work is by [5], who—partially supported by Samsung—hinted that the garbage collection functional-ity allows the drive controller to introspect the file alloca-tion table of known file systems (i.e., NTFS, ext3 and ext4) to autonomously decide which blocks can be safely wiped, without the OS intervention or the implementation of TRIM. As a matter of fact, in Sect. 5.2 we document our tests on

the garbage collection functionality under these hypotheses to see whether the results from various drives could confirm its presence and behavior. Our experiment lead to results that contradict [5]'s work.

4.2 Write caching in SSD experiments

SSDs are equipped with a small amount of DRAM-based cache memory, which is used to reduce the number of physical writes. This feature must be taken into account when performing experiments on SSDs, because it affects the number and speed of writes.

For instance, biases introduced by caching were not addressed by [4], who performed experiments by writing graphic and text files on an SSD and then verified that they were completely unrecoverable after deleting them. Although in [4] there is no clear statement about the OS used, in 2008 no OS had TRIM support for SSD drives [3], and therefore file deletion could not be a consequence of TRIM. The only explanation is that the files were not large enough to completely fill the drive cache (usually around 512 MB to 1 GB), and were therefore never actually written on disk: they were simply erased from cache, and no trace was left to allow a full or partial recovery. The same bias can be noted in [15], where the percentage of recoverable blocks when using small files is considerably lower than the same percentage with big files, even under the same conditions and usage patterns. This can be explained by the fact that small files get usually stored in cache and not written to disk.

Disabling write caching via software drivers or via the OS (e.g., via the `hdparm -W 0` command), while recommended, does not always succeed. The only reliable way to avoid being biased by caching is to use large files for write operations, in order to fill up the cache and force the drive to physically write on flash cells.

5 Implementation details

We apply the methodology described in this section on three SSDs, each with a different controller and combination of features, namely a Corsair F60 (controller: SandForce SF-1200), a Samsung S470 MZ-5PA064A (controller: Samsung ARM base 3-core MAX) and a Crucial M4 (controller: Mar-vell 88SS9174-BLD2). As stated by the vendors, these drives implement wear leveling. Furthermore, the Corsair F60 per-forms data compression, whereas both Samsung S470 and Corsair F60 are said to implement garbage collection. Table 1 summarizes the functionalities according to the official spec-ifications. Instead of presenting the results of our tests in a separate section, for the sake of practicality we explain the results immediately after the description of each step.

Table 1 SSD features as reported by vendors

SSD	WL	TRIM	GC	Compression
Corsair F60	✓	✓	✓	✓
Samsung S470	✓	✓	✓	
Crucial M4	✓	✓		

5.1 TRIM

Using trimming, whenever blocks are erased or moved on an SSD, they are added to a queue of blocks that should be blanked. This operation is lazily performed by the garbage collector process as soon as the disk is idle, preparing blocks to be (re)written and ensuring balanced read-write speeds. Trimming needs operating system support (it should be noted that even where supported it is not always enabled by default), as the OS needs to tell the controller when a block can be trimmed. Windows 7 and 8 (and Server 2008R2), as well as Linux version 2.6.28 and later support trimming.

5.1.1 Methodology

Figure 1 shows the steps required to determine whether and how an SSD implements trimming. Before start, the disk is wiped completely and the write cache is disabled or filled, as detailed in Sect. 4.2. Then, a stub filesystem is created and filled with random content (i.e., files) up to different percentages of their capacity: 25, 50, 75 and 100 %. This is because certain controllers exhibit different trimming strategies depending on the available space. The tests described below are repeated for each level of filling.

As both [15] and our experiments show that some TRIM implementations behave differently when dealing with quick formats or file deletions, we analyze both cases indepen-dently.

When a quick format command is issued, the OS will sup-posedly notify the SSD that the whole drive can be trimmed. The disk is left idle and the filesystem is checked for zeroed blocks. If the SSD implements TRIM procedures, we expect to observe changes in the number of zeroed blocks; other-wise, no changes will happen. To check for zeroing, we use a sampling tool—as [5] did—which loops over the entire disk and samples 10 Kb of data out of every 10 MB. It then checks whether the sample is completely zeroed. Whenever a non-zeroed sample is encountered, the tool checks whether it is zeroed in subsequent loops. The sampling size was chosen empirically to find a good trade-off in terms of overhead and accuracy. The choice of the sampling size depends on the time that the analyst wants to spend on this test, and only affects the final decision. The test ends when the situation does not change within a timeout, which can be sometimes obtained

Fig. 1 TRIM test flow

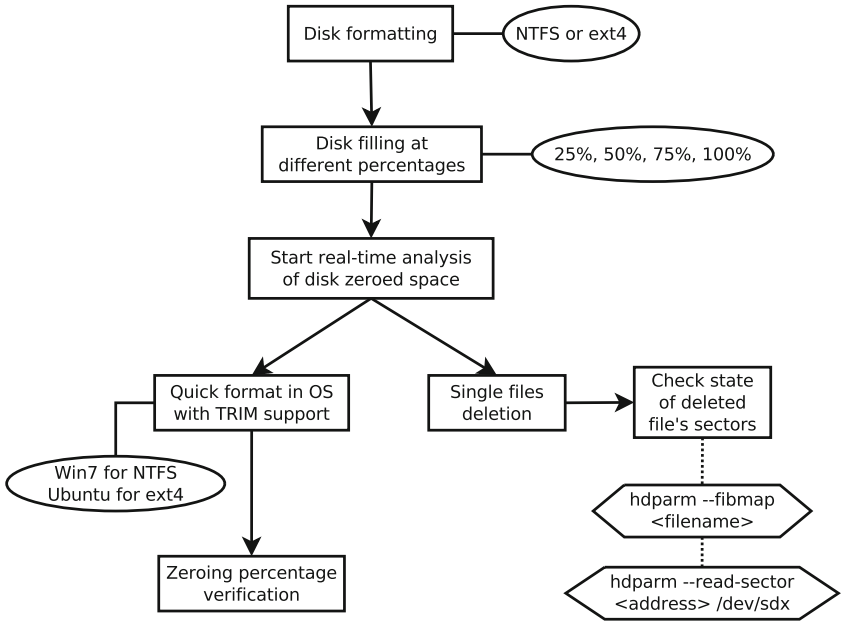
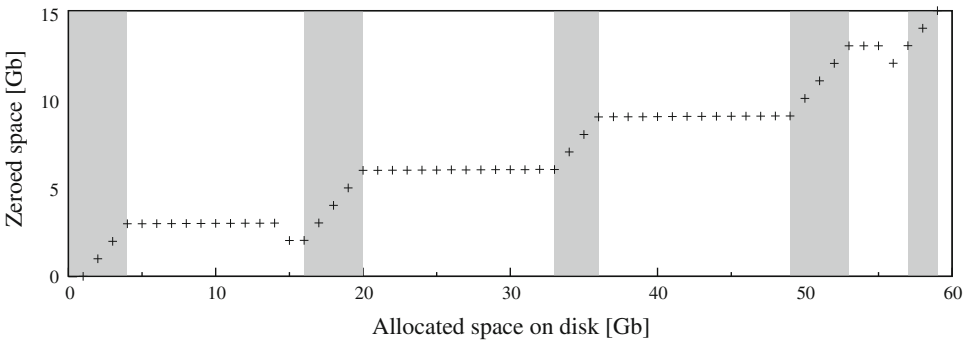


Fig. 2 The amount of blocks erased by TRIM in our Corsair F60 disk depends on the amount of used space



by the vendor's documentation (i.e., the time between cycles of execution of the garbage collector). If no documentation is available, we set a very long timeout (i.e., 24 h, based on several trials that we run on all our disks). During our experiments, we found out that if the TRIM functionality is present and active, it triggers within 1–10 s.

To test the behavior for deletion, we proceed in a similar way. Our workflow deletes single files from the filesystem and monitors their respective blocks. Depending on the file size, sampling may not be necessary in this case. The test ends, as before, when all the (remaining) portions of the erased file do not change within a timeout.

5.1.2 Results

We run this test on Windows and Linux, which both included stable support for TRIM.

On NTFS (Windows 7), Samsung S470 and Crucial M4's trimming was very aggressive in both quick format and file deletion: The disk was (apparently, see below for discussion) wiped in 10 s by the Samsung S470 controller; on the Crucial M4, wiping occurred even before notifying the OS. Similar results were obtained with file deletion: the sectors were completely wiped in 5 and 10 s respectively. The Corsair F60, instead, behaved differently. After issuing a quick format, only a small percentage of data was erased; when we repeated the test at different filling levels, we surprisingly found out that the fraction of erased blocks is someway proportional to the total used space, as shown in Fig. 2: There are some thresholds that define how much space must be trimmed depending on the used space. In particular, there are 5 ranges in which the amount of zeroed space increases linearly, whereas it remains constant at all the other filling values. The Corsair F60 behaved unexpectedly also when dealing with file deletions: Some files were wiped in at most 3 s after deletion, whereas some other files were not wiped at all and could be recovered easily, depending on their allocation. This discovery spurred an interesting study of the erasing patterns, which is explained separately in Sect. 5.3.

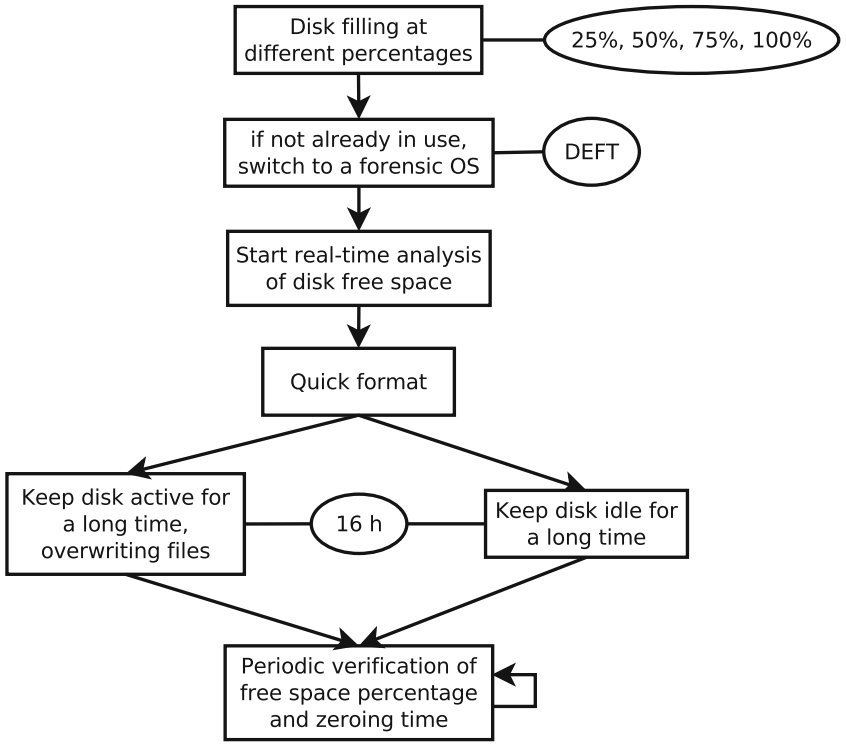


Fig. 3 Garbage collection test flow

On ext4 (Ubuntu Linux 12.04) we obtained significantly different results. In the quick-format branch the outcomes are similar across different disks: The entire content of the SSD was (apparently) erased in about 15 s. This can be explained by the fact that for all the SSDs Linux used the same AHCI device driver. The single file deletion, instead, showed a different behavior. The Samsung S470 did not erase any block and all the files were completely recoverable. The Crucial M4 apparently did not erase any file, at least until the device was unmounted; at that point the blocks were erased. Apparently, the driver notifies a file deletion only when it becomes absolutely necessary to write data on disk (i.e., when the disk is unmounted or when the system is in idle state long enough to flush data on the non-volatile storage). The Corsair F60 showed none of the behaviors exhibited with NTFS: All the files were erased correctly. Supposedly, Windows drivers implement a different trimming policy, or the SandForce controller used by this SSD features NTFS-specific optimizations.

In the paragraphs above, we often noted that the files, or the drives, were "apparently" erased within 10–15 s. The reason for using this word is the following. While checking for the zeroing, we are reading the disk through the controller (as our methodology is purely black-box). Therefore, we cannot be sure if the memory packs have been physically zeroed or if the controller returns a content of zeros for any block which is tagged for trimming, regardless of its physical content.

5.2 Garbage collection

5.2.1 Methodology

The entire test must be carried out on an OS and drivers that do not support TRIM, as the effects of the two features would not be distinguishable from a black-box viewpoint. Figure 3 summarizes our test to determine whether GC is implemented and when it starts.

After the usual preliminary steps, the dummy filesystem is created and filled with random files. The content is not important, yet the size is: we must use large files due to the considerations in Sect. 4.2. Then, the same sampling procedure described in Sect. 5.1 is started, and a quick format is issued. As there is no reliable information regarding the trig-gering context and timeout, our methodology explores two different paths. First, the disk is kept idle to allow the trig-gering of the GC. Alternatively, the SSD is kept active by continuously overwriting existing files, adding no new con-tent. [5] found out that GC triggers in almost 3 min. Some non-authoritative sources, however, state that a reasonable timeout ranges between 3 and 12 h. Our methodology pro-poses to wait up to 16 h before concluding the experiment.

5.2.2 Results

Even hours after the default timeout, none of the SSDs performed GC. Since Samsung S470 and Corsair F60 were advertised as having GC capabilities, we devised a simple

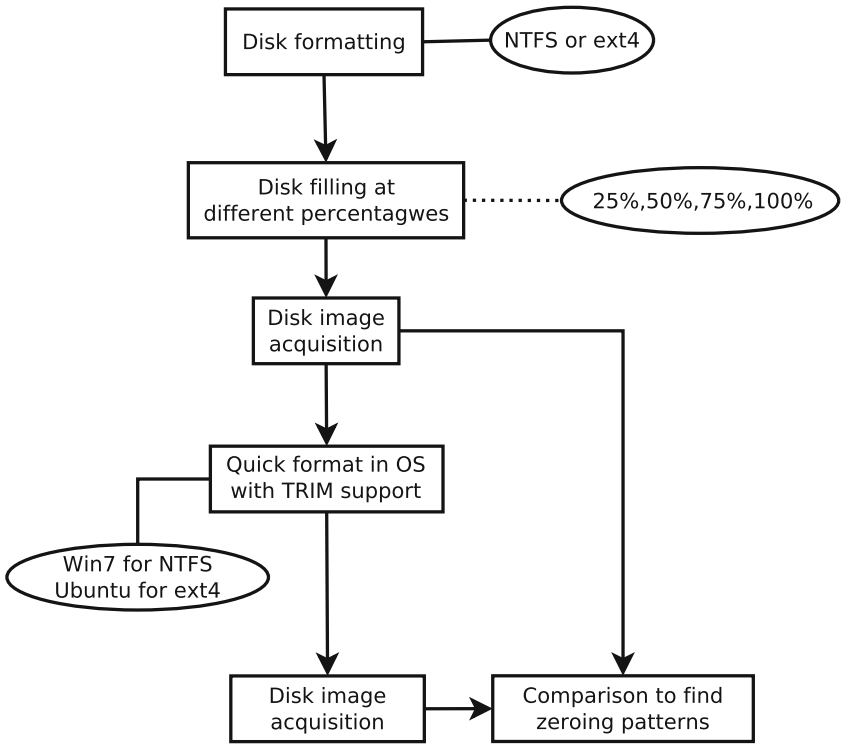


Fig. 4 Erasing patterns test flow

additional test to validate this result. This goal was to determine what percentage of non-random files can be recovered after a quick format. We filled each disk with the same JPEG image until there was no space left on the device, and formatted it on a TRIM-incompatible OS and let it idle. As shown in Table 5, even with simple tools (e.g., Scalpel), we recovered 100 % of the files from both drives, confirming that no GC occurred. Note that we used a carver merely as a baseline for recoverability: Our approach is not meant to evaluate the performance of carvers in general.

5.3 Erasing patterns

As showed in Sect. 5.1, certain SSD controllers (e.g., Corsair F60 with NTFS) may exhibit unexpected trimming patterns. Therefore, we devised a workflow to further explore these cases and assess to what extent a forensic acquisition is affected by TRIM or GC.

5.3.1 Methodology

As shown in Fig. 4, after the preliminary steps the disk is filled with a dummy filesystem containing files with random data (the content is irrelevant as we are focusing on how the controller handles deletion). Then, a raw image of the disk is acquired with `dd` before issuing a quick format instruction. A second raw image is then acquired after a while (see the considerations in Sect. 5.1 on timeouts). Clearly, "raw" here refers to what the controller exposes to the OS. The obtained images are compared block-wise to highlight erased sections. This test must be ran at different disk filling levels, in case the controller behaves differently based on the amount of free space.

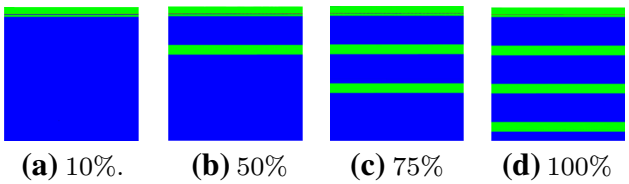


Fig. 5 Test results for erasing patterns test performed on Corsair F60 SSD: at different filling levels an increasing number of evenly-spaced stripes are visible. *Green areas* are zeroed by the controller, while *blue areas* remain unchanged. The *non-erased blocks in the first stripe* (**a**) contain the copy of the master file table and are therefore not zeroed (color figure online)

5.3.2 Results

We applied this test on the Corsair F60 SSD, which exhibited odd behavior in the TRIM test. We analyzed it at 10, 50, 75 and 100 % of space used. At each level we analyzed the entire disk as explained above and, using custom scripts, mapped disk sectors into the maps shown in Fig. 5. Interestingly, we notice four stripes in predictable areas (green) where the files are surely going to be erased, whereas the rest of the disk (blue) is not modified even after file deletion. The small difference in the first stripe is due to the fact that the master file table is allocated within it, and this portion is not erased (Fig. 5a).

Table 2 File-recoverability test for Corsair F60 SSD

Position	Written	Recovered	%
Within erased stripes	29,545	1	0.34
Outside erased stripes	71,610	71,607	99

Only one of the files that were written within the erased areas could be recovered, whereas 99 % of those outside those bounds could be retrieved with standard tools

Table 3 Compression ratio of the files used for compression test

File	gzip	7zip	bz2	Entropy
`/dev/zero`	1,030	7,086	1,420,763	0.0
`/dev/urandom`	1.0	0.99	0.99	0.99

Files with high (Shannon's) entropy are difficult to compress and there-fore result in more data to be written on disk

This result is consistent with the results described in Sect. 5.1. In particular, the first four linearly-increasing por-tions that appear in the TRIM test result of Fig. 1 correspond to the very same green areas highlighted with this erasing-pattern experiment. Therefore, the reaction of the controller when dealing with file deletion on NTFS is explainable by the very same green areas: If a file is allocated within the green stripes, it will surely be erased by TRIM, whereas files that fall outside the green areas are not trimmed.

We validated this result as follows. We formatted the drive and filled it with easily-recoverable files (i.e., JPEG image files, as the JPEG header is easy to match with carvers). Then, we selectively deleted the files allocated inside or outside the green stripes and, after acquiring the entire disk image, tried to recover them, and to map their (known) position against the stripes position. Table 2 shows that only 0.34 % of the files within the erased stripes are recovered, whereas this percentage reaches 99 % for files allocated entirely outside the green areas, thus confirming the results of Fig. 5.

5.4 Compression

5.4.1 Methodology

The key intuition behind this test is that the overhead due to hardware compression is negligible in terms of time. Thus, it will take considerably less time to physically write a compressed file with respect to an uncompressed one. However, this could not be the case if the controller actually goes back and compresses the files afterward as a background task.

As shown in Table 3, compression algorithms yield the best results with low-entropy files, whereas are not very effective on high-entropy data. We therefore created two differ-ent files with very different levels of entropy: `/dev/zero` and `/dev/urandom`. The methodology is summarized in Fig. 6. After creating the two files (10 Gb each, to bypass write caching), we monitor via `iostat` the time spent in transfer and the throughput: A high throughput indicates compression, as less data is physically written on disk.

5.4.2 Results

As Fig. 7a shows, both file transfers in the Samsung S470 took almost the same time, showing no sign of compression.

The Crucial M4 test showed in Fig. 7b yielded the same results, even if with different values. This drive exhibits systematic performance glitches at the same points in time for each run. This happens almost every 25 s, regardless of the file's size and transfer time, and does not happen with other drives under the same conditions. We transferred files of different sizes and under different conditions (i.e., computer, source, OS) but obtained consistent results. We can speculate that these glitches are due to some computations performed by the controller at regular time intervals.

The Corsair F60 is the only one advertised as having com-pression capabilities. Indeed, as shown in Fig. 7c, it behaves in a very different fashion: The transfer time for compress-ible files is about one third files than for incompressible files. Therefore, we can infer that the actual amount of data physically written on disk is considerably lower, meaning that the controller compresses it transparently.

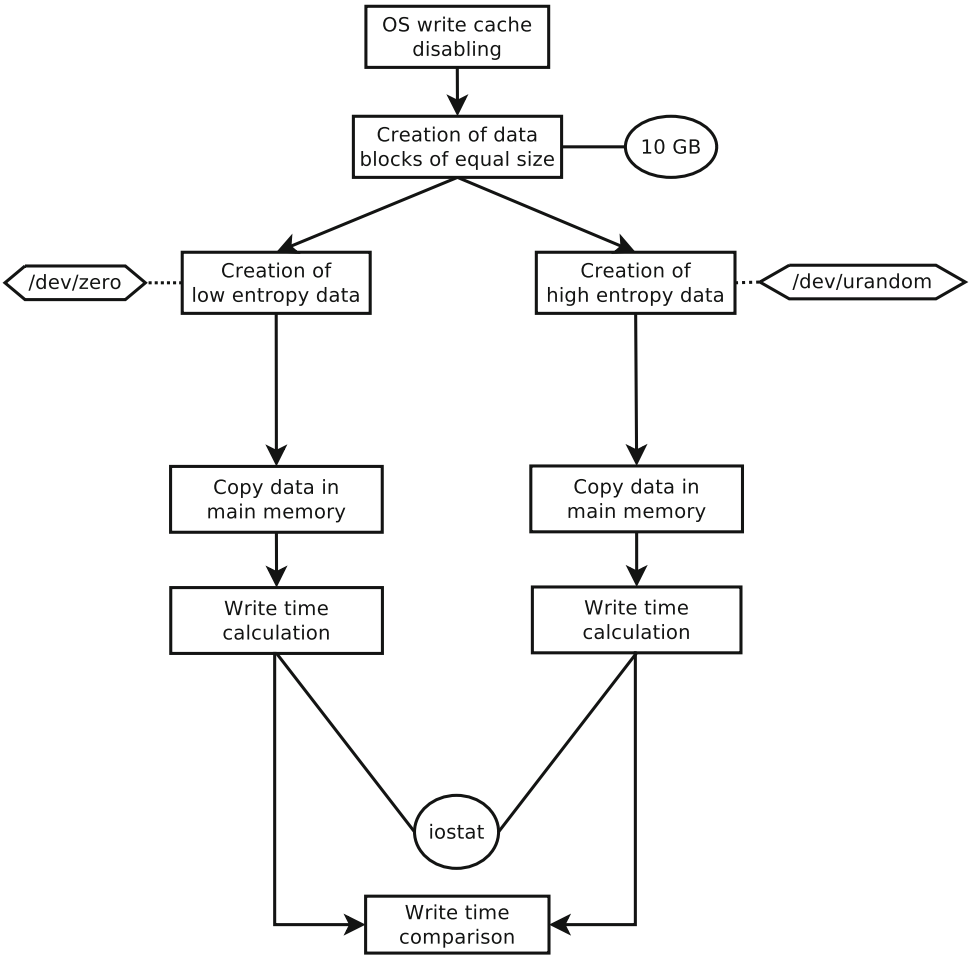
All graphs show an initial transitory phase with a very high transfer rate, due to the effect of write caching on the initial portion of the files being sent to the disk, as explained in Sect. 4.2. However, the effect of caching does not affect our results.

5.5 Wear leveling

Wear leveling is a very common and basic feature, but none of the examined vendors clearly states what happens to the old versions of the same file (i.e., how write amplification [13] is treated). From a black-box viewpoint there are two possible situations. One alternative is that old blocks are not erased and remain where they were: in this case a carver may be able to extract many different versions of the same file, rep-resenting a clear snapshot of the data at a given point in time. Alternatively, the old data may be erased, moved out of the addressable space or simply masked by the controller, which in this case would tell the OS that no data is present (virtually zeroed block). Unfortunately, if we get no data from the disk, there is no way (with a black-box approach such as ours) to determine which of these is the case.

Another detail to take into account when dealing with wear leveling is that vendors do not explicitly reveal the conditions under which the functionality is triggered. From the available information and previous work ([1,2,10]) it appears that two conditions must hold: there should be enough free

Fig. 6 Compression test flow



blocks with a lower write count than the one that is being overwritten, and there must be a certain difference between the writes of the used block and the ones of the new destination block. Since the precise values depend on the vendors and are not publicly available, in our experiments we erred on the side of caution and left at least 25 % of the disk capacity free, and overwrote the same blocks more than 10,000 times to cover whatever write cycle gaps may be present.

5.5.1 Methodology

Our test is not aimed at determining if an SSD implements a wear leveling feature, since this is pretty much standard nowadays. From the forensic viewpoint, what matters is if wear leveling can be leveraged via black-box analysis to recover data. If a drive has no wear leveling capabilities, or if write amplification is completely masked by the FTL, the end result is the same: nothing is lost and nothing is gained from a forensic viewpoint.

Our test flow is shown in Fig. 8. As the entire test flow requires the continuous re-writing of the same files, and it is extremely important that these write operations are physically sent to the disk, the considerations in Sect. 4.2 are of paramount importance.

The disk is filled up to 75 % with a dummy filesystem. Since wear leveling is internal, the file or filesystem type has no impact, so we choose files with known patterns (to ease carving operations afterward), and an ext4 filesystem under Ubuntu Linux.

At this point files are overwritten with new data (of exactly the same size) a total of 10,000 times, while monitoring the amount of zeroed space on disk. Should it decrease, it means that the controller wrote the new data in a different, less used position, leaving the old data intact. To validate this, if the control on zeroed space is positive, carving can be used to determine if different versions of the same files are effectively recoverable.

5.5.2 Results

We ran our test on all the disks in our possession. Our results confirmed our expectations, yielding negative results. As we mentioned, we cannot know what the controller is really

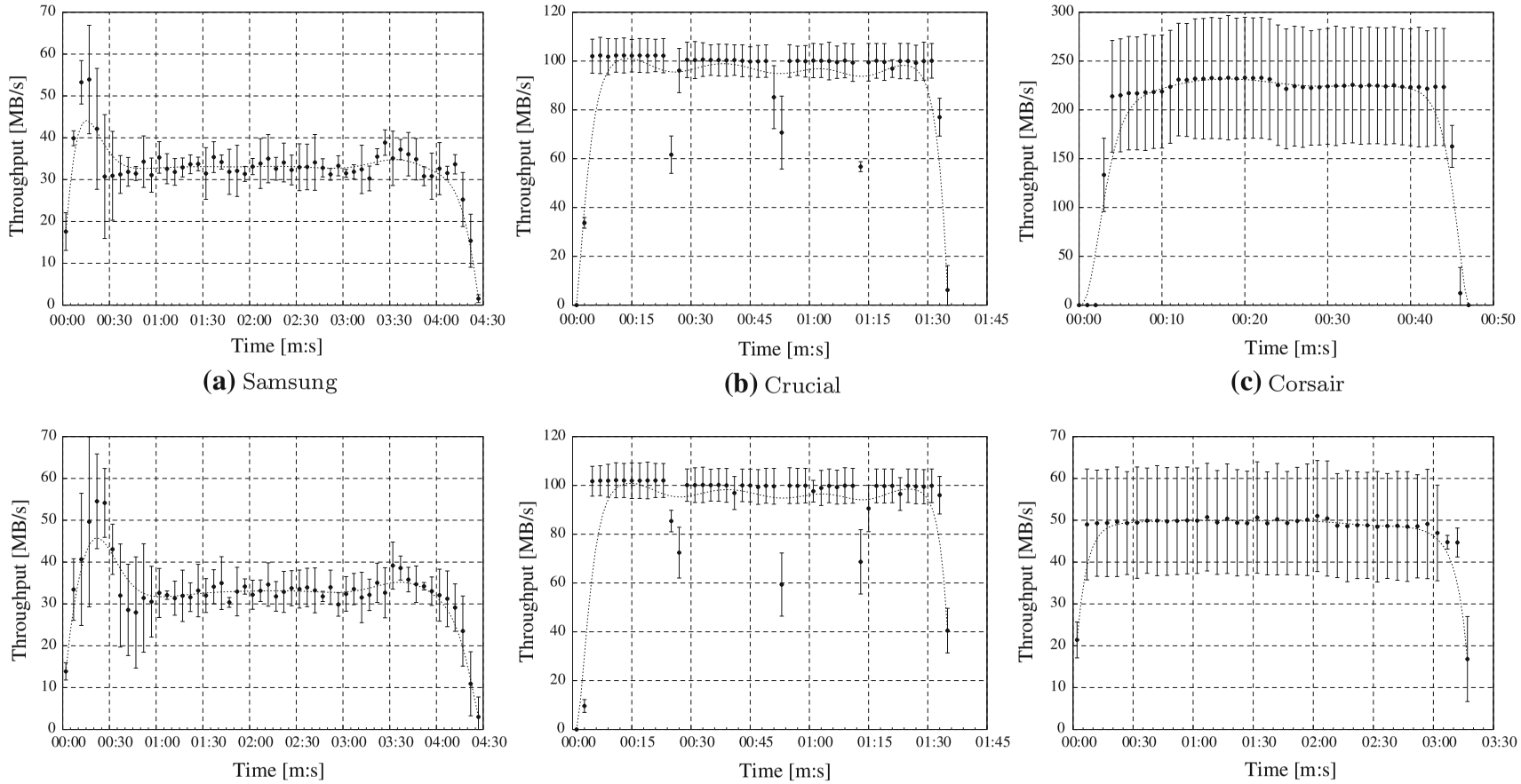
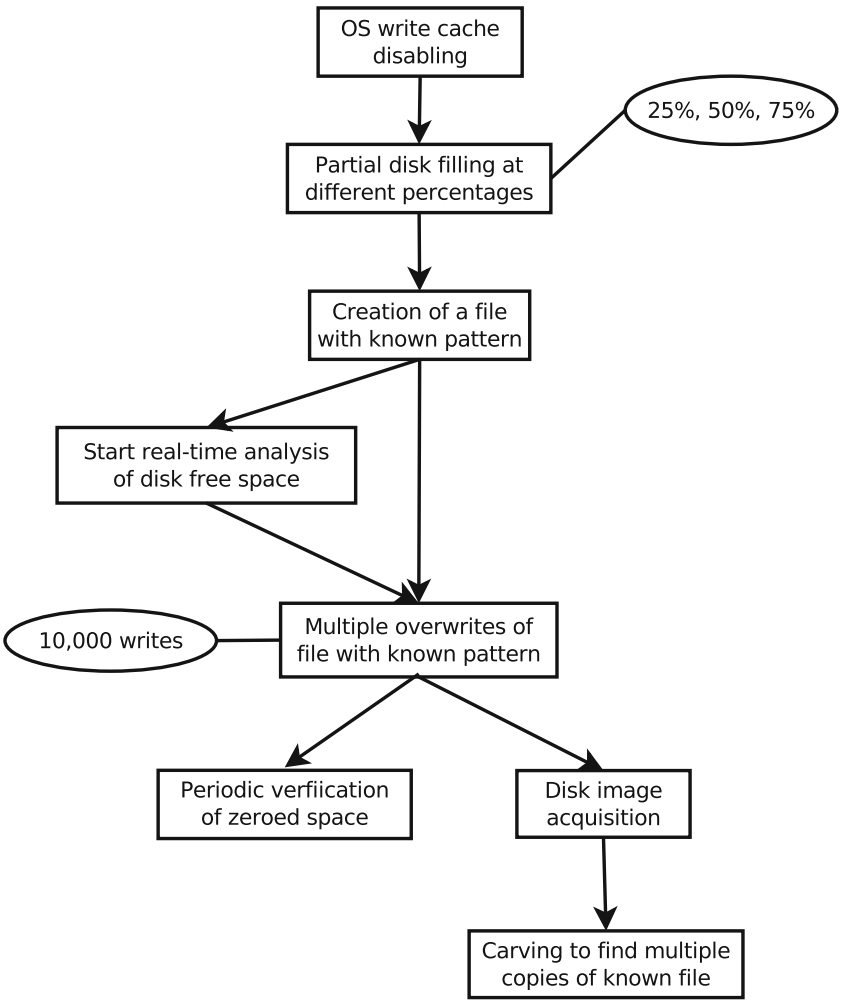


Fig. 7 Mean and variance of the sampled throughput among 15 repeated transfers of 10 GB low and high-entropy files (top and bottom row, respectively). For **a** and **b** low and high-entropy file transfers have almost the same shape and duration, showing that the controller does not perform any kind of optimization (i.e., compression) on data before writing it. On the other hand, in **c** throughput with low-entropy files is considerably higher and the entire file transfer takes less than 1/3 than the high-entropy files transfer. This result confirms that less data had to be physically written on disk, which means that compression was indeed performed by Corsair drives

Fig. 8 Wear leveling test flow



doing behind the scenes; what we know is just that the effects of write amplification are completely masked by the controller, and thus a standard forensic procedure will not be impacted by wear leveling.

5.6 Files recoverability

The tests described so far are all aimed at determining if some functionalities implemented by a given SSD are forensically disruptive, to ultimately allow a forensic analyst to assess whether some data is still retrievable. What usually interests the forensic analyst most, however, is being able to access and retrieve files on an SSD much in the same way as on a traditional HDD.

The tests that we propose in this section determines how much an SSD behaves similarly to a HDD from a data-recoverability viewpoint. In HDDs, recoverability is affected by the filesystem policies on overwriting previous data. In SSDs, in addition to this, trimming, garbage collection, and the other unexpected controller behavior described so far negatively impact the recoverability.

5.6.1 Methodology

The test flow is shown in Fig. 9. The drive is first initialized with a dummy filesystem and filled with "carver-friendly" files: In our case, we wrote JPEG files of around 500k each, and then quick formatted the drive. After the usual 24 h timeout, we used Scalpel to attempt a file recovery.

5.6.2 Results

We ran our experiment on all our disks with a NTFS filesystem with enough copies of the same JPEG image to fill the entire drive. As summarized in Table 4, both the Crucial M4 and the Samsung S470 have a zero recovery rate, which means that the TRIM functionality tested in Sect. 5.1 actually works and erases all of the deleted files.

The Corsair F60 behaves differently, as shown in Sect. 5.3: 71,607 files out of the 101,155 were recovered, totaling a 70.79 % recovery rate on NTFS. Curiously, all the files that were only partially recovered—or not recovered at all—were all contiguous in small chunks. On Ext4, instead, TRIM did not allow the recovery of any file.

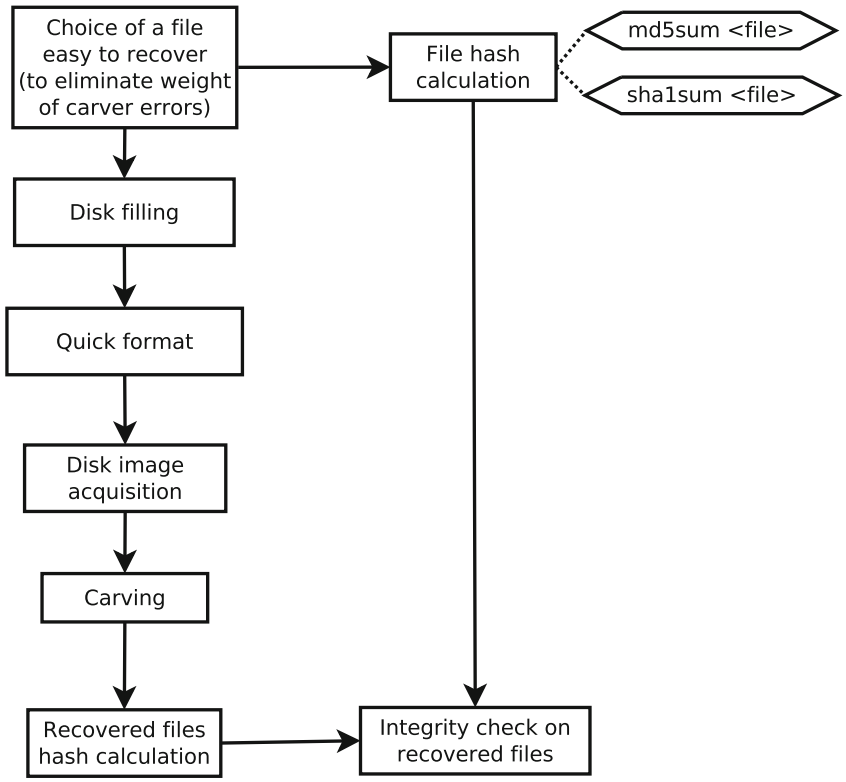


Fig. 9 Files recoverability test flow

Table 4 Files recoverability test results

SSD	FS	Written	Recovered	%
Samsung	NTFS	112,790	0	0
	ext4	110,322	0	0
Corsair	NTFS	101,155	71, 607	70.79
	ext4	99,475	0	0
Crucial	NTFS	112,192	0	0
	ext4	110,124	0	0

The drives implementing an aggressive version of TRIM (Samsung S470 on NTFS and Crucial M4), did not allow the recovery of any file after a format procedure. The Corsair F60 on NTFS, as expected, has a non-null recovery rate due to the erasing pattern its TRIM implementation exposes. On ext4, however, this same disk allowed the recovery of 0 out of 99,475 files

6 Use cases

We applied our methodology and in this section we provide two use cases, a test-driven triage classification of drives according to forensic friendliness, and the development of an anti-forensic technique specifically suited to a given drive.

6.1 Ranking drives

Although proposing a comprehensive and accurate classification of SSDs goes beyond the scope of this paper, we show how our methodology can be applied to indicate "forensic friendliness" of an SSD. We consider the output of the TRIM, GC and File Recoverability tests. We follow the workflow exemplified in Fig. 10.

- **A. HDD Equivalent.** The SSD behaves as a HDD. Standard forensics tools are expected to work as usual. SSDs in this class present no disruptive behaviors (e.g., TRIM, GC).
- **B. High Recoverability.** TRIM and other wiping functionalities are implemented but they are not very aggressive: an HDD-equivalent recovery is expected.
- **C. Low Recoverability.** SSD functionalities are quite aggressive and succeed in deleting or masking most of the deleted data that could have been recovered from a HDD. It is, however, still possible to achieve some results with standard tools.
- **D. Complete Wiping.** No deleted data can be recovered using standard black-box tools. White-box analysis may be a solution but it is not guaranteed to yield acceptable results. This is the worst possible case when performing a forensic analysis on a SSD.

Applying this method to our drives we obtained the following classification. The Crucial M4 implements a very effective TRIM functionality with any filesystem which directly makes the recoverability test yield a 0 % rate so, even though the garbage collector does not trigger, the associated class is *D. Complete wiping*: this drive is very likely to make recov-

Table 5 Files recoverability without TRIM on Samsung S470 and Corsair F60 drives

SSD	Written	Recovered	%
Samsung	1,12,790	1,12,790	100
Corsair	1,01,155	1,01,155	100

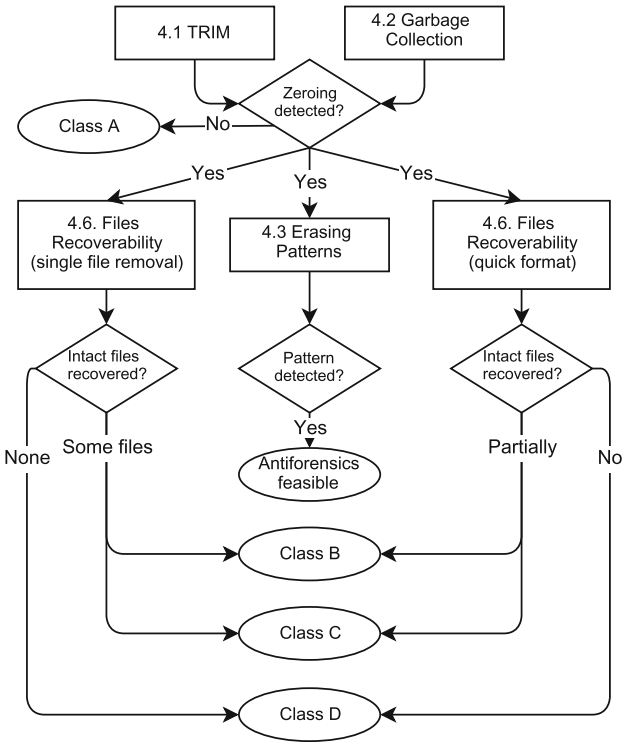


Fig. 10 Use case workflow for assessing the forensic friendliness of a SSD

Table 6 Test results for the disks in our possession and derived classification

SSD	FS	TRIM	GC	Recover (%)	Class
Samsung S470	NTFS	Full	×	0	D
	ext4	Full	×	0	D
Crucial M4	NTFS	Full	×	0	D
	ext4	Full	×	0	D
Corsair F60	NTFS	Partial	×	70.79	B
	ext4	Full	×	0	D

ery of deleted files impossible. The same happens on the Samsung S470 with both NTFS and ext4 filesystems and on the Corsair F60 with ext4. Corsair F60 SSD with NTFS, instead, present only a partially working TRIM implementation, which allows the recovery of almost 71 % of deleted files; this combination between drive and filesystem is therefore associated to a *B. High recoverability* class. These results are summarized in Table 6.

6.2 An antiforensic filesystem

In Sect. 5.3 we discussed our results on the different behaviors of different trimming implementations.

We also discussed how we found out that the Corsair F60 drive behaves weirdly when formatted with NTFS and subjected to a quick format command. Only a small percentage of data is physically erased (as opposed to complete deletion, which is what happens on other TRIM enabled drives). Repeating the test at different filling levels, we found the per-centage of erased blocks to be somehow proportional to the percentage of used space on disk (as shown in Fig. 2).

By filling the drive with easily-recoverable files, and then selectively deleting them inside or outside the erased stripes, we obtained the results in Table 2, which shows that only 0.34 % of the files within certain specific areas of the disk can be recovered, while this percentage reaches 99 % for files allocated entirely outside said areas (which in the following we will call “green areas”, according to the color used in the graphs, for simplicity).

This peculiarity can be used for antiforensic purposes, i.e. to make sure that certain files of interest will be physically wiped from the drive after any deletion command by the trimming algorithms.

An easy possibility to make use of this would be to position a 4 GB-sized container (such as a mountable disk image) in the stripes. In this case, a simple delete command would ensure complete destruction of the image, even if the hard drive were unplugged during, e.g., a search and seizure oper-ation. Alternatively, smaller files that need to be securely wiped out after use could be stored in these portions of the drive.

We investigated the feasibility of exploiting this behavior by implementing a simple user-space filesystem that allocates a given file in the green areas. We realized our proof-of-concept, called S2D2 (https://bitbucket.org/necst/s2d2) in Python. It supports the following operations:

- initialize the working area of disk, cycling over the green areas, and creating a boot sector and an appropriate, simple metadata structure
- maintain the metadata for stored files, which for simplicity we implemented by mean of a Python list, holding a tuple of values for each file (i.e., path, first sector, size).
- create files and write them to disk, adding padding to ensure that the file size is multiple of 512 bytes, creating an entry as appropriate in the file list, and writing the data in the green area.
- read files, which is straightforward
- delete files, implemented by issuing a trim command to the underlying controller.

In addition, we structured the prototype to allow to allocate and manage files on the blue areas, to be able to compare the

effect. As a reminder, those areas behave like a conventional HDD, with no evidence of trimming support.

We evaluated the effectiveness of our antiforensic filesys-tem prototype assessing the recoverability of a file once deleted. We did not evaluate the speed of our filesystem implementation, as this is not our focus and in any case per-formant userspace filesystems exist [17].

As expected, all the files written in the green areas were wiped by the controller when marked as deleted by the OS, while all other files were not physically deleted from the SSD's Flash chips. The same behavior was found when issu-ing a quick format procedure instead of a file deletion: the FAT on disk was reset, but the only files actually wiped were the ones in the green areas.

A forensic analyst, finding and being able to retrieve deleted files on the rest of the drive, would easily be fooled into thinking that the controller does not implement trimming, especially if a quick format procedure had been previ-ously issued and everything (except wiped files) was retrieved via carving. Applying thoroughly our methodology, they would be able to know in advance of this possible attack. It is worth noting that even in our proof-of-concept implementation the remainder of the file system is completely consistent.

Besides the specific attack shortly presented here, which is targeted to one specific controller, we think that this use case exemplifies a whole new class of vulnerabilities (namely, abuse of controller characteristics and unexpected behaviors) that will arise thanks to the growing diffusion of SSDs, and which our methodology allows to discover in new drives.

7 Future directions

Although our blackbox workflow and experimental results are far more complete than what has been proposed in previous work, a few areas remain for further investigation and research.

7.1 Firmware dependency

Each SSD comes with its own firmware version, which basi-cally embeds (part of) the FTL logic. As such, it determines the SSD characteristics and, therefore, its forensics "friend-liness" with respect to the features tested by our workflow. Extending our methodology to take into account the firmware revision is challenging for two reasons: First, not all SSD vendors release firmware upgrades. Second, and most impor-tantly, firmware upgrades are often one-way procedures (i.e., the only way to downgrade a firmware to a previous version would consist in buying another SSD, provided that the old version of the firmware is still on the market); this affects the repeatability of the experiments in a scalable way.

7.2 OS dependency

The triggering of TRIM depends on the specific combination of OS, filesystem type, device driver, and AHCI commands implemented. The current version of our workflow explores the OS and filesystem type. For instance we have shown in our experiments how the Corsair behaves differently under Windows (NTFS) and Linux (ext4). In a similar vein, future extensions of our work should consider other variables such as the device driver and AHCI commands.

Another important dependency that should be explored is that the forensics examiner needs to know the OS version before performing an investigation. The availability of is this contextual information varies from case to case, and due to the sensitive nature of forensic cases there are no statistics on this.

In summary, future work should focus on running our methodology on a wide range of OSs, SSD brands and models, in order to create a reference catalog useful for forensic investigations.

8 Conclusions

In order to overcome the intrinsic limitations of SSDs, onboard controllers adopt a number of advanced strategies, preemptively erasing blocks of deleted files (possibly even if not solicited by the OS) and even performing compression or encryption on the data. Consequently, SSDs cannot be treated as standard HDDs when performing a forensic analysis: Standard tools and well-known techniques are based on the assumption that the hard drive does not modify or move data in any way, that every block can be read if it was not previously wiped, and that reading a block will yield the data physically contained by that block.

Each vendor implements a different controller, and therefore each SSD acts differently. We proposed a complete testing methodology and applied it to three drives of leading vendors. For each test we interpreted the results to provide the forensic analyst a practical way to fine tune their approaches to the acquisition of SSD drives, which can be very similar to acquisition of HDDs, or completely different. Indeed, we showed that the combination of controller, OS, filesystem and even disk usage can deeply influence the amount of information that can be retrieved from a disk using forensic procedures. Thanks to our methodology, we were able to investigate the peculiarities of each implementation (e.g., a drive selectively wiping blocks only in specific portions of it). We also showed how this peculiarity could be easily exploited to obtain antiforensic delete operations.

We also proposed a test aimed at verifying the implementation of a compression feature on the SSD. We showed that our test can indeed devise whether the drive uses compression

or not; this is useful to determine the feasibility of a deeper white-box approach. We also investigated the controversial topic of garbage collection to see under what conditions it triggers.

Each of the proposed experiments is tailored to inspect how each functionality works and how "aggressive" it is. They allow the analyst to know, for example, if the SSD has a disruptive TRIM implementation or if GC does not work under particular conditions. All these information can be of great help when performing a forensic acquisition on such drives, since they can change the expectations or even suggest some particular techniques to adopt in order not to allow the controller to delete possible useful data. Also, the result can help the forensic expert to estimate the success of costly procedures such as a memory white-box analysis.

Acknowledgments This paper was published, in an earlier version [7], in the proceedings of the ACSAC 2013 conference. The authors are grateful to the anonymous reviewers and to the conference atten-dees, who pointed out weaknesses and significantly contributed to the improvement of this research. We wish to particularly acknowledge the fruitful discussions at the conference with Dr. Sarah Diesburg. The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007–2013) under Grant Agreement No 257007, as well as from the TENACE PRIN Project (No. 20103P34XC) funded by the Italian Ministry of Education, University and Research.

References

1. Wear leveling in micron NAND flash memory. Tech. Rep. TN-29-61, Micron Technology Inc. (2008). http://www.micron.com/~/media/Documents/Products/Technical
2. Wear-leveling techniques in NAND flash devices. Tech. Rep. TN-29-42, Micron Technology Inc. (2008). http://www.micron.com/~/media/Documents/Products/Technical
3. Microsoft Co.: Support and Q&A for Solid-State Drives. MSDN Blog (2009). http://blogs.msdn.com/b/e7/archive/2009/05/05/support-and-q-a-for-solid-state-drives-and.aspx
4. Antonellis, C.J.: Solid state disks and computer forensics. ISSA J. **6**(7), 36–38 (2008)
5. Bell, G.B., Boddington, R.: Solid state drives: the beginning of the end for current practice in digital forensic recovery? J. Digit. Forensics Secur. Law **5**(3), pp. 1–20 (2010)
6. Billard, D., Hauri, R.: Making sense of unstructured flash-memory dumps. In: SAC '10, pp. 1579–1583. ACM, New York (2010)
7. Bonetti, G., Viglione, M., Frossi, A., Maggi, F., Zanero, S.: A comprehensive black-box methodology for testing the forensic characteristics of solid-state drives. In: Proceedings of the Annual Computer Security Applications Conference (ACSAC). ACM (2013). doi:10.1145/2523649.2523660
8. Breeuwsma, M., De Jongh, M., Klaver, C., Van Der Knijff, R., Roeloffs, M.: Forensic data recovery from flash memory. Small Scale Digit. Device Forensics J. **1**, 1–17 (2007)
9. Bunker, T., Wei, M., Swanson, S.: Ming II: a flexible platform for NAND flash-based research. Tech. Rep. CS2012-0978, UCSD CSE (2012)
10. Chang, Y.H., Hsieh, J.W., Kuo, T.W.: Improving flash wear-leveling by proactively moving static data. IEEE Trans. Comput. **59**(1), 53–65 (2010)
11. Diesburg, S., Meyers, C., Stanovich, M., Mitchell, M., Marshall, J., Gould, J., Wang, A.I.A., Kuenning, G.: Trueerase: per-file secure deletion for the storage data path. In: Proceedings of the 28th Annual Computer Security Applications Conference, ACSAC '12, pp. 439–448. ACM, New York (2012). doi:10.1145/2420950. 2421013
12. Gray, J., Fitzgerald, B.: Flash disk opportunity for server applications. Queue **6**(4), 18–23 (2008)
13. Hu, X.Y., Eleftheriou, E., Haas, R., Iliadis, I., Pletka, R.: Write amplification analysis in flash-based solid state drives. In: SYSTOR '09, pp. 10:1–10:9. ACM, New York (2009)
14. Intel: AP-684: Understanding the flash translation layer (FTL) specification. Intel Application Note. (1998) http://www.jbosn.com/download_documents/FTL_INTEL.pdf
15. King, C., Vidas, T.: Empirical Analysis of Solid State Disk Data Retention When Used with Contemporary Operating Systems, pp. S111–S117. Elsevier Science Publishers B. V., Amsterdam (2011)
16. Luck, J., Stokes, M.: An integrated approach to recovering deleted files from nand flash data. Small Scale Digit. Device Forensics J. **2**(1), 1941–6164 (2008)
17. Rajgarhia, A., Gehani, A.: Performance and extension of user space file systems. In: Proceedings of the 2010 ACM Symposium on Applied Computing, SAC '10, pp. 206–213. ACM, New York (2010). doi:10.1145/1774088.1774130
18. Skorobogatov, S.P.: Data remanence in flash memory devices. In: Cryptographic Hardware and Embedded Systems—CHES 2005, 7th Intl. Workshop, Edinburgh, UK, August 29–September 1, 2005, Proc., Lecture Notes in Computer Science, vol. 3659, pp. 339–353. Springer, Berlin (2005)
19. Templeman, R., Kapadia, A.: Gangrene: exploring the mortality of flash memory. In: HotSec'12, pp. 1–1. USENIX Association, Berkeley (2012)
20. Wei, M., Grupp, L.M., Spada, F.E., Swanson, S.: Reliably erasing data from flash-based solid state drives. In: FAST'11, pp. 8–8. USENIX Association, Berkeley (2011)