Politecnico di Milano

SCUOLA DI INGEGNERIA DELL'INFORMAZIONE
Corso di Laurea Magistrale in Ingegneria Informatica

UN APPROCCIO SISTEMATICO ALL'ANALISI FORENSE DI DISCHI A STATO SOLIDO

Autori:
Gabriele Bonetti (766390)
Marco Viglione (748965)

Relatore:
Prof. Stefano Zanero
Correlatori:
Dr. Alessandro Frossi
Dr. Federico Maggi

Anno accademico 2011-2012

Sommario

L'introduzione su larga scala di supporti di memorizzazione a stato solido ha portato con sé la necessità di rivedere le pratiche di analisi forense fino ad oggi utilizzate su dischi magnetici rotazionali. Le tecnologie a stato solido introducono infatti una serie di nuove funzionalità, tali da necessitare la definizione di specifiche metodologie di testing che prendano in considerazione le peculiarità di questi dispositivi. La complessità nella definizione di un tale protocollo di analisi risiede nella necessità di operare su vari livelli, software e hardware, e di approcciare una disciplina nuova, in rapida evoluzione e in buona parte inesplorata. In questo lavoro affrontiamo il problema della definizione di un protocollo di testing che consideri le opportunità e le problematiche introdotte dai dischi a stato solido e della sua implementazione software per automatizzarne i processi.

Indice

Elenco delle figure

Elenco delle tabelle

1. Introduzione

La tecnologia alla base dei supporti di memorizzazione di massa, rimasta pressochè inalterata per oltre un decennio, ha subito negli ultimi anni un'evoluzione importante, dovuta all'introduzione su larga scala di supporti di memoria a stato solido. Le metodologie di analisi forense, ormai consolidate nell'estrazione di dati da supporti di memoria basati su tecnologie magnetiche, non hanno però subito un'altrettanto rapida evoluzione. Le procedure di analisi comunemente utilizzate, infatti, non considerano l'elevata complessità delle logiche di gestione dei dischi a stato solido, risultando perciò meno complete e rigorose in termini forensi. Pochi tentativi sono stati fatti per adattarle a questa nuova tecnologia, tutti con notevoli limiti. Lo stato dell'arte è attualmente fermo ad un'analisi puramente quantitativa della recuperabilità dei dati, senza alcun tentativo di studiare dettagliatamente il comportamento dei controller dei dischi, responsabili di gran parte delle complessità aggiuntive. La principale conseguenza è che nessuno dei lavori finora svolti si è occupato di investigare, se non superficialmente, le condizioni di attivazione delle funzionalità peculiari dei dischi e i criteri secondo cui esse operano. Per questo motivo consideriamo importante definire una metodologia di analisi dei dischi e di estrazione dei dati che tenga conto delle peculiarità tecnologiche che le memorie a stato solido hanno portato con sè.

Vista l'assenza in questo ramo di ricerca di una metodologia di testing completa, il primo contributo che il nostro lavoro intende fornire è una proposta per tale metodologia. In particolare abbiamo definito nel dettaglio le singole procedure di testing necessarie ad analizzare la presenza e gli effetti di ogni tecnologia implementata nei dischi a stato solido che possa influire sul recupero dei dati, fornendo anche un'implementazione software per eseguire in modo automatico i test proposti. Il secondo contributo consiste nel proporre una metrica di classificazione dei dischi sulla base della loro analizzabilità forense, calcolabile attraverso i risultati dei test da noi proposti. Vorremmo così fornire, ad analisti forensi e piu in generale ad aziende che gestiscono dati sensibili, una metrica per valutare il livello di recuperabilità, o dualmente di sicurezza, dei dati nei dischi a stato solido. Infine abbiamo applicato la metodologia proposta sui dischi in nostro possesso, effettuando le analisi delineate teoricamente. Per eseguire tali test abbiamo sviluppato

sia i singoli tool necessari per effettuare le analisi, sia degli script capaci di effettuare in modo automatizzato i singoli esperimenti. Laddove possibile abbiamo inoltre studiato come i risultati ottenuti possono essere sfruttati da un punto di vista antiforense, sviluppando gli strumenti necessari ad approfittare delle vulnerabilità trovate per ostacolare il recupero di specifici dati.

La tesi è strutturata nel modo seguente:

Nel *Capitolo 2* introduciamo i temi dell'informatica forense e delle procedure di acquisizione sicura dei dati utilizzate per dischi rotazionali. Presentiamo poi le tecnologie di archiviazione dei dati basate su memorie a stato solido, le loro caratteristiche peculiari e le soluzioni tecniche utilizzate per massimizzare prestazioni e vita utile dei dispositivi, evidenziando le differenze rispetto ai dischi magnetici in termini prestazionali, di consumo energetico e di funzionamento. Nel corso del capitolo indichiamo inoltre come le funzionalità peculiari dei dischi a stato solido possano teoricamente aiutare od ostacolare l'analisi di tali dispositivi. Infine analizziamo lo stato dell'arte della ricerca sull'analisi forense di tali dispositivi, evidenziando contributi e limiti dei lavori esistenti.

Nel *Capitolo 3* presentiamo dettagliatamente i protocolli di testing creati e le metriche utilizzate per definire l'analizzabilità forense dei dischi. Per ogni protocollo sono illustrate le motivazioni che ci hanno portato a definire ogni passo, le scelte implementative che abbiamo adottato ed il suo contributo nella classificazione dei dischi secondo la metrica proposta.

Nel *Capitolo 4* le metodologie proposte vengono applicate sperimentalmente ai dischi in nostro possesso, utilizzando i software da noi creati per la loro automatizzazione. Per ogni esperimento discutiamo gli strumenti utilizzati e i risultati ottenuti, fornendo delle linee guida per la loro interpretazione. Infine illustriamo un esempio di come l'esito dei test ci permette di classificare i dischi utilizzati secondo le metriche di analizzabilità proposte.

Nel *Capitolo 5* traiamo le conclusioni del nostro lavoro, discutendo dei contributi originali che abbiamo fornito e proponendo sviluppi futuri per continuare la ricerca nell'ambito dell'analisi forense di memorie a stato solido.

Infine in *Appendice* forniamo il codice dei tool da noi realizzati e degli script di automatizzazione dei test.

2. Analisi forense di dischi a stato solido

In questo capitolo introduciamo i concetti essenziali dell'analisi forense dei dispositivi di memorizzazione dati, evidenziando le differenze fra tecnologie magnetiche tradizionali e tecnologie a stato solido. I supporti di memorizzazione a stato solido presentano infatti alcune caratteristiche peculiari nei processi di lettura e scrittura, oltre ad un limite al numero di letture e scritture possibili, causate dai fenomeni fisici alla base di questa tecnologia. Vedremo quindi quali funzionalità sono state introdotte dai produttori di memorie a stato solido per superare questi problemi e garantire alte prestazioni e un arco di vita almeno pari a quello di un disco magnetico.

2.1 L'informatica forense

Con il termine *informatica forense* si intende la ricostruzione di eventi o fonti di prova, a partire dai dati, mediante tecniche scientifiche ripetibili. Al termine *scientifico* è infatti associato, sin dai tempi di Galileo, il concetto di *ripetibilità*. Un esperimento di analisi forense deve essere scientifico, cioè strutturato in modo tale che sia possibile effettuare il medesimo test su una copia dei dati, ottenendo i medesimi risultati. Questo richiede che l'esperimento non sia descritto solo attraverso input e output, ma anche attraverso una documentazione dettagliata della procedura seguita e della logica alla base di ogni operazione. Nell'esecuzione di procedure forensi, l'uso di strumenti di analisi open source è da considerarsi preferibile poichè consente, grazie alla disponibilità del codice sorgente, di comprendere le procedure che il software esegue, consentendo almeno in linea teorica la reale ripetibilità delle operazioni e rendendo così trasparente ogni azione eseguita.

Nel contesto legale, il concetto di ripetibilità assume inoltre un significato molto specifico: accertamento ripetibile significa, secondo l'articolo 359 del codice di procedura penale, un'analisi che non causi alterazione della prova. Nell' ambito dell'informatica forense questa definizione introduce un problema particolarmente delicato, a causa delle caratteristiche peculiari delle prove digitali. Esse sono infatti intrinsecamente fragili: ciò

significa che se un dato digitale viene alterato, non c'è modo di rilevare tracce dell'alterazione, ed è quindi possibile la creazione un "falso perfetto". É dunque necessario seguire precise procedure di acquisizione e di analisi capaci di assicurare l'identità, l'autenticità e l'integrità delle prove digitali durante ogni fase dell'investigazione, rendendole robuste contro le alterazioni.

2.2 L'analisi forense di dischi magnetici

Con la crescente diffusione di dispositivi digitali verificatasi nell'ultimo decennio, la necessità di analizzare dispositivi di memorizzazione digitali è diventata sempre più presente in ambito investigativo e processuale. Sono quindi state definite una serie di procedure di acquisizione ed analisi standard, con lo scopo di garantire la validità dell'elemento di prova in sede legale.

Un disco rotazionale è composto da una pila di piatti rotanti su cui una testina mobile si posiziona per effettuare operazioni di lettura o scrittura. Come mostrato in figura 2.1 dati vengono memorizzati sulla superficie magnetica dei piatti in strutture chiamate tracce, ovvero cerchi concentrici situati sulla superficie del disco. I piatti sono divisi in settori e i settori sono a loro volta raggruppati in cluster, che rappresentano la più piccola area del disco indirizzabile dal sistema operativo.

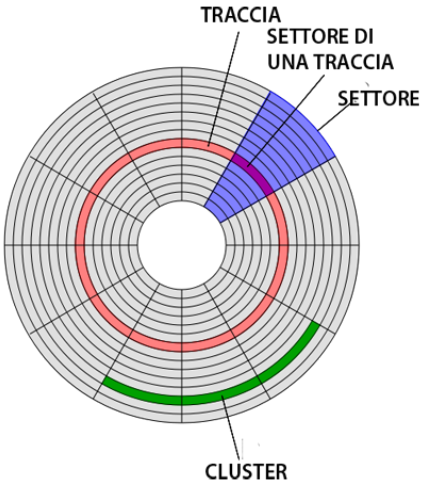


Figura 2.1: Geometria interna di un HDD

I movimenti meccanici della testina introducono diverse latenze, rotazionali e di posizionamento, dovute sia alla ricerca del piatto su cui operare sia al posizionamento sulla traccia richiesta sia all'attesa del passaggio di un settore specifico sotto la testina. Per

minimizzare l'impatto di queste significative latenze sulle prestazioni complessive del sistema, i sistemi operativi allocano i dati secondo principi di località e persistenza, che possono essere sfruttati per il recupero dei dati.

In aggiunta, in caso di cancellazione dei file senza azzeramento dei singoli settori occupati, il sistema operativo si limita a marcare il file come cancellato nella tabella di allocazione dei file del file system, rendendone possibile il recupero. Il contenuto di un file viene effettivamente cancellato solo nell'istante in cui i settori da esso utilizzati vengono sovrascritti, fatto che potrebbe avvenire anche molto tempo dopo. Inoltre, essendo il cluster la più piccola area indirizzabile dal sistema operativo, se il nuovo dato sovrascritto non occupa l'intero cluster negli ultimi settori rimarrà uno spazio, chiamato *slack space* (vedi Figura 2.2), contenente ancora parte dei dati precedenti, che potrebbero essere rimasti inalterati. Non sempre questi frammenti saranno significativi (pensiamo ad esempio a file eseguibili o compressi, decifrabili solo se recuperati nella loro interezza), ma per alcuni formati di file contenenti semplice testo tali frammenti potrebbero essere sufficienti per recuperare in forma comprensibile almeno parte dei dati.

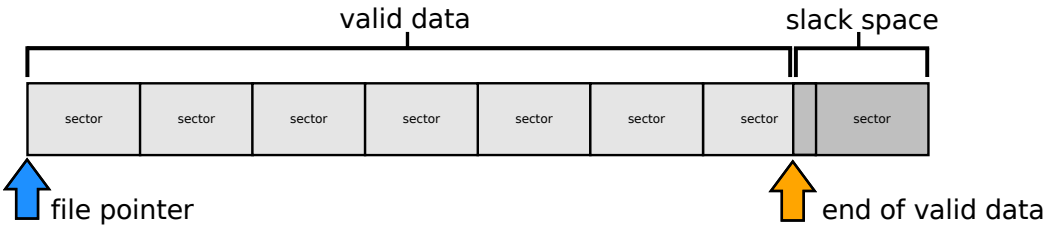


FIGURA 2.2: Stato di un cluster dopo una sovrascrittura.

2.2.1 La procedura di acquisizione forense

La comune procedura di acquisizione forense di un disco rotazionale è suddivisa nei seguenti passi:

- *Valutazione stato del sistema*: se il dispositivo a cui i dischi sono collegati è spento, si può procedere staccando i dischi dalla macchina. Se il dispositivo è invece acceso, bisogna considerare varie opzioni per non compromettere dati importanti che potrebbero essere recuperabili solo a macchina attiva. É buona norma procedere nell'acquisizione dei dati in ordine decrescente di volatilità, partendo cioè dall'acquisizione della memoria primaria, proseguendo con lo stato del sistema, i suoi processi e connessioni di rete, per poi concludere con le memorie di massa.
  É molto importante, nello spegnimento della macchina, evitare di passare attraverso la normale procedura di arresto del sistema operativo. Durante tale procedura,

infatti, esso effettua delle operazioni di scrittura (come la chiusura di file o l'aggiornamento del file di journaling) che potrebbero alterare lo stato della prova. La procedura migliore è quindi staccare l'alimentazione, documentando tutte le operazioni eseguite.

- *Calcolo dell'hash*: è una procedura che consente, attraverso la computazione di una funzione di hashing (md5, sha1), di creare una impronta digitale dello stato corrente dei dati contenuti nel disco. Il calcolo dell'hash su un disco da acquisire deve avvenire quanto prima, poichè l'hash è in grado di garantire l'integrità della fonte di prova soltanto a partire dal momento in cui viene calcolato per la prima volta ed è fondamentale per garantire l'integrità dei dati sul disco durante tutta la procedura di investigazione.

- *Acquisizione dell'immagine del disco*: il passo successivo consiste nell'acquisire una copia esatta bit a bit del disco, utilizzando software e hardware che garantiscano la non alterazione dei dati durante la copia. In particolare è necessario utilizzare un sistema operativo specificatamente studiato per l'analisi forense, generalmente basato su Linux, progettato in modo da evitare qualunque operazione di scrittura sul dispositivo (ad es. Helix, DEFT). Per una maggiore sicurezza si utilizza di norma anche un dispositivo hardware chiamato *write blocker*, che impedisce al disco di ricevere comandi di scrittura dal sistema operativo, così da tutelare l'analista da errori involontari o da scritture causate da processi non sotto il suo controllo.
  In particolari condizioni in cui non sia possibile l'estrazione fisica dei dischi e la loro acquisizione individuale o in cui siano presenti sistemi RAID che renderebbero complessa la ricostruzione dei dati, è possibile avviare, sulla macchina da analizzare, *distribuzioni live* (che si avviano cioè senza la necessità di installazione) dei sistemi operativi per uso forense già citati. Questa soluzione deve essere per quanto possibile evitata, poichè l'assenza di un write blocker espone maggiormente al rischio di alterazione dei dati.
  Infine l'immagine va acquisita utilizzando strumenti come *dd*, *dcfldd* (che ha il vantaggio di calcolare l'hash del disco durante la copia) o *dc3dd*. A copia conclusa è necessario calcolare l'hash della copia e confrontarlo con l'hash del disco originale, per confermare che la copia sia identica.

- *Conservazione della prova*: sia il disco originale che la prima copia vanno conservati in condizioni capaci di garantirne identità, autenticità e integrità. Per analizzare il disco verranno effettuate ulteriori cloni della copia primaria, da utilizzare come copie di lavoro. Alcuni strumenti di analisi possono infatti alterare irrimediabilmente i dati nel tentativo di recuperarli o ricostruirli.

Una volta che si ha a disposizione una copia su cui lavorare, è possibile effettuare molte operazioni utili ai fini dell'analisi forense, come il recupero di file cancellati, di partizioni formattate o compromesse, l'analisi dello slack space, l'accesso ai bad block o il recupero di dati da dispositivi danneggiati. Tuttavia, come vedremo a breve, questa procedura ormai consolidata per i dischi magnetici non può essere applicata altrettanto efficacemente su memorie a stato solido.

2.3 Struttura ed analizzabilità delle memorie a stato solido

Per comprendere quali problemi causano le memorie a stato solido nel corso delle acquisizioni forensi è importante descrivere la struttura e le peculiarità di tali memorie. Esistono diversi tipi di memorie a stato solido. Questo capitolo descrive gli aspetti essenziali delle memorie flash, concentrandosi su quelle di tipo NAND, che sono le più diffuse per la costruzione di dischi a stato solido. In particolare viene illustrata sia la loro struttura fisica che le funzionalità implementate per ottimizzare tali memorie al fine di utilizzarle nelle periferiche di archiviazione, evidenziando quali di esse ne influenzano l'analizzabilità forense. In particolare:

- Azzeramento preventivo delle celle in background (Sezione 2.3.3.1)
- Presenza di copie multiple dei file su disco (Sezione 2.3.3.5)
- Esecuzione prevalente di scritture sequenziali (Sezione 2.3.3.1)
- Cifratura a livello di controller (Sezione 2.3.5)
- Compressione a livello di controller (Sezione 2.3.3.1)

2.3.1 Le memorie flash

La tecnologia alla base dei dischi a stato solido è quella delle memorie flash [1]. La memoria flash è un tipo di memoria non volatile che può essere programmata elettronicamente. Esistono due diverse tipologie di memorie flash, NOR e NAND, che differiscono nello schema di connessione degli array di memoria di cui sono costituite e nella modalità di indirizzamento utilizzata nelle operazioni di lettura o scrittura. Le memorie NOR sono versatili, possono essere utilizzate sia per contenere codice da eseguire che piccole quantità di dati; le memorie NAND invece sono ottimizzate per lo storage di grandi quantità di dati.

Le memorie flash di tipo NOR furono sviluppate inizialmente come alternativa alla tecnologia EPROM (Erasable Programmable Read Only Memory), con lo scopo di avere latenze di accesso ridotte e permettere l'esecuzione di codice senza la necessità di caricamento in memoria primaria. Le celle sono collegate in parallelo, permettendo di leggere o programmare ogni cella individualmente. Il metodo di accesso è casuale come nelle RAM: la memoria può essere letta byte per byte in tempo costante. Vengono usati BUS separati per indirizzamenti, letture e scritture. Questo tipo di memoria flash è ideale per contenere BIOS e firmware, informazioni che raramente necessitano di aggiornamenti.

Nelle memorie flash di tipo NAND, al contrario, le celle sono collegate in serie, impedendo la lettura o scrittura individuale di ciascuna di esse: tali operazioni possono essere applicate a non meno di un'intera serie di celle interconnesse, solitamente equivalente a un blocco di 512 byte. Il BUS è condiviso tra operazioni di indirizzamento e di trasferimento di dati e, a causa delle sue dimensioni ridotte (8 o 16 bit) può richiedere tra 3 e 5 cicli di clock per trasmettere un indirizzo. Rispetto alle memorie NOR, le NAND possono memorizzare più informazioni in uno spazio minore, richiedono un tempo inferiore per la cancellazione ed il costo per bit è molto inferiore. Grazie alla densità di storage e al basso consumo di energia, questo tipo di memorie è comunemente usato in vari dispositivi portatili come telefoni cellulari e fotocamere digitali. Anche la totalità dei dischi a stato solido analizzati nel corso di questo lavoro utilizza questo tipo di memoria flash.

	NOR	**NAND**
Capacità	1MB - 32MB	16MB - 512MB
XIP (Esecuzione di codice)	Si	No
Performance	Cancellazione lenta (5 s)	Cancellazione veloce (3 s)
	Scrittura lenta	Scrittura rapida
	Lettura rapida	Lettura rapida
Cicli di Cancellazione	10,000 - 100,000	100,000 - 1,000,000
Metodo di Accesso	Casuale	Sequenziale
Interfaccia	Memory mapped (simile a SRAM)	I/O mapped (accesso in burst da 512 byte)
Prezzo	Alto	Basso

Tabella 2.1: Differenze principali tra memorie flash di tipo NOR e NAND

2.3.2 Le memorie NAND

Le singole celle NAND sono composte da un transistor di tipo floating gate, completamente circondato da materiale isolante (per garantire la non volatilità della memoria) e comandato da un control gate.

Per la scrittura e la cancellazione delle celle, si utilizzano due diversi fenomeni fisici chiamati rispettivamente *tunnel injection* e *tunnel release* [2]. Poichè il gate è l'unica connessione tra transistor e word line, in seguito a una scrittura una carica negativa resta intrappolata al suo interno. Tali elettroni fungono da barriera tra floating gate e control gate, e quindi quando la carica che passa attraverso il gate viene letta il risultato è 0. Tale risultato è il valore logico associato a una cella scritta.

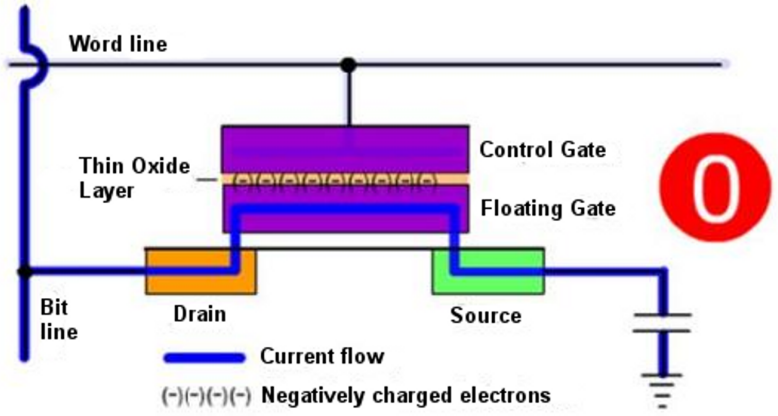


FIGURA 2.3: Stato di una cella NAND programmata. Gli elettroni intrappolati tra floating gate e control gate formano una barriera. Il valore di lettura della carica passante nel gate è quindi 0.

Durante il processo di cancellazione, invece, la carica negativa intrappolata viene forzata ad attraversare il materiale isolante fino a giungere in superficie. Senza la barriera di carica negativa, la lettura della carica passante attraverso il gate supererà una determinata soglia, restituendo come valore logico della cella cancellata 1 [3].

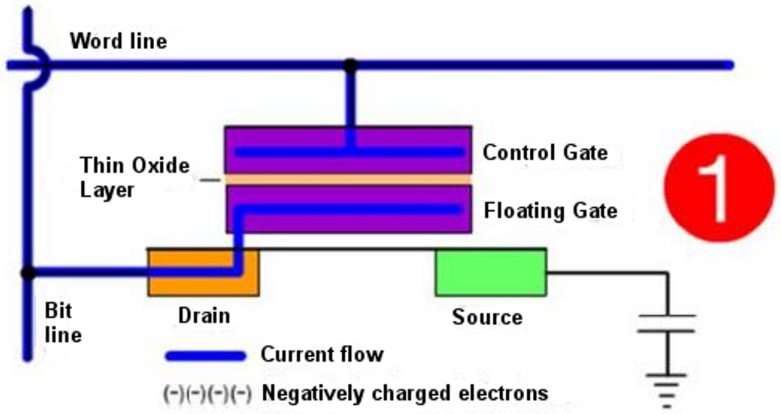


FIGURA 2.4: Stato di una cella NAND cancellata. Senza alcuna barriera tra floating gate e control gate, il valore di lettura della carica passante nel gate è 1.

Le memorie flash conservano i dati all'interno di array di transistor floating gate. Esistono due tipi di memorie NAND, single level cell (SLC) e multi level cell (MLC), studiate per classi diverse di applicazioni [4].

- I transistor SLC possono assumere soltanto due stati, 0 e 1 quindi sono grado di memorizzare un unico bit. La capacità ridotta e i costi più elevati sono compensati da un più rapido tempo di scrittura, una maggiore durata e un error rate inferiore. Sono l'ideale per applicazioni che richiedono alta affidabilità e buone performance.
- I transistor MLC invece possono invece assumere un numero maggiore di stati, solitamente 4, consentendo di memorizzare più bit di dati in una singola cella. Alcuni vantaggi immediati consistono nella maggiore capacità e densità rispetto ai SLC. Tuttavia la finestra di tensione usata è la stessa degli SLC, nonostante il maggior numero di stati, quindi la distanza di separazione tra i diversi livelli di tensione è più piccola. Questo causa un più alto error rate (poichè è più facile che un valore cambi a causa dell'effetto di disturbi), performance di lettura/scrittura ridotte dovute alla maggiore accuratezza richiesta, un aumento nel consumo di potenza, minor data retention rate e un ridotto tempo di vita delle celle.

Il nome delle memorie NAND deriva dalla loro struttura di connessione, simile a quella delle porte NAND (vedi Figura 2.5). I transistor sono connessi in serie e solo se tutte le word line sono a valore logico alto la tensione della bit line si abbassa.

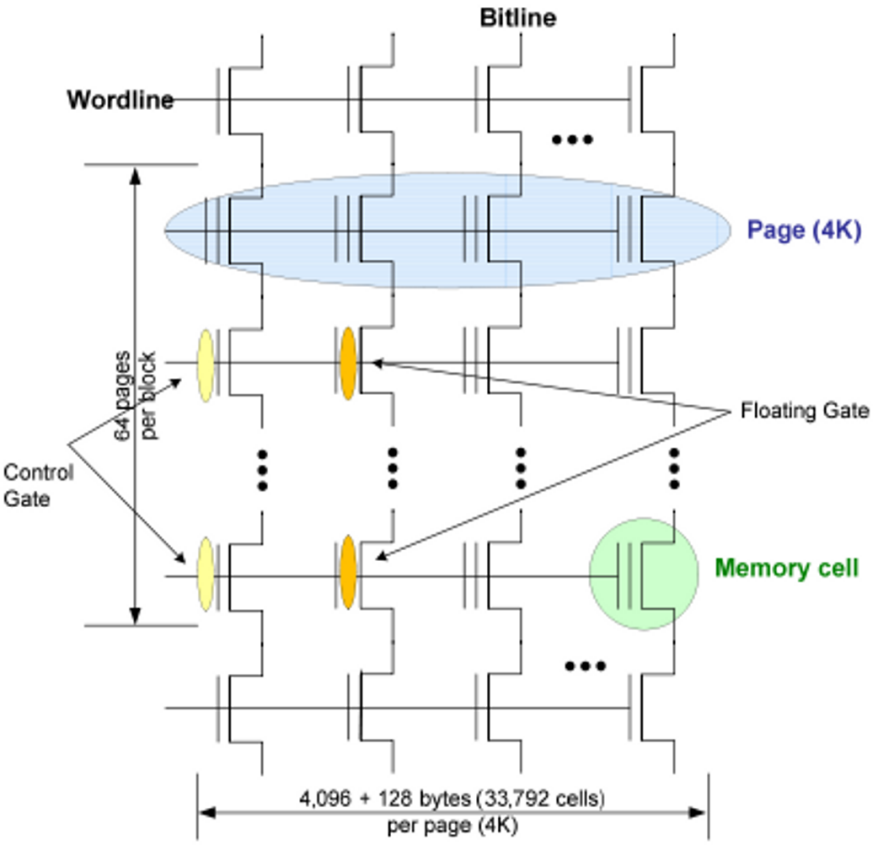


FIGURA 2.5: Schema di connessione dei singoli transistor in un'architettura flash NAND (8Gb SLC 50 nm). La connessione è in serie.

2.3.3 Caratteristiche tecnologiche delle memorie a stato solido

I dischi a stato solido posseggono una serie di peculiarità tecnologiche e di problemi differenti dai dischi tradizionali, per affrontare i quali sono state implementate diverse tecnologie che rendono un disco a stato solido un dispositivo molto più complesso di un disco magnetico. Le memorie flash NAND, in particolare, hanno alcune caratteristiche intrinseche che con l'aumentare della densità (e la diminuzione delle dimensioni) dei transistor diventano generazione dopo generazione sempre più evidenti. I problemi più noti sono: [4]

- Necessità di cancellare una cella prima di riscriverla
- Azzeramenti costosi in termini energetici e prestazionali
- Errori di lettura e scrittura causati da disturbi
- Problemi di data retention
- Necessità di gestire un elevato numero di bad block
- Numero molto limitato di scritture per cella (10.000 - 100.000)

Negli anni sono state proposte molte soluzioni. Vediamo ora le più diffuse e importanti.

2.3.3.1 Cancellazioni e sovrascritture

Il tempo di cancellazione di una memoria NAND è relativamente lungo, poichè il blocco è la più piccola unità cancellabile e una cella non può essere sovrascritta senza essere preventivamente azzerata. Inoltre, se un blocco contiene frammenti di più file, ogni volta che uno di questi file viene cancellato è necessario riscrivere i frammenti degli altri file altrove prima di effettuare l'azzeramento. Una simile moltiplicazione dei dati da scrivere (definita *write amplification*) oltre ad allungare il tempo necessario all'operazione potrebbe ridurre l'arco di vita del dispositivo, quindi vengono adottate varie contromisure.

Una soluzione piuttosto comune consiste nell'implementare algoritmi di distribuzione dei dati su disco che prediligano l'esecuzione di scritture sequenziali. Se si riuscisse a fare in modo che il contenuto dei blocchi sia associato ad un solo file, infatti, non vi sarebbe la necessità di eseguire queste scritture aggiuntive. Da un punto di vista dell'analisi forense questo faciliterebbe anche il lavoro dei carver, aumentando le possibilità di recuperare in forma integra i dati.

Molte altre contromisure tendono tuttavia ad ostacolare l'analizzabilità del disco. Una di queste, finora poco diffusa, consiste nell'implementare la compressione automatica dei dati a livello di controller. Sebbene analizzando il dispositivo attraverso i dati visibili a un sistema operativo (analisi black box) questa funzionalità agisca in modo completamente trasparente, nel momento in cui si volessero invece acquisire direttamente i contenuti delle memorie scavalcando il controller (analisi white box) i dati ottenuti potrebbero risultare illeggibili.

Un'altra soluzione, presente nelle specifiche di molti dischi, consiste nell'implementazione sempre a livello di controller di un algoritmo di garbage collection che si occupi in background (solitamente mentre il disco è in stato di idle) di effettuare preventivamente operazioni di azzeramento o fusione di blocchi sotto-utilizzati. Poichè il blocco è la più piccola unità cancellabile, il garbage collector si occupa anche di trasferire le pagine ancora valide in una zona libera della memoria. Dal punto di vista dell'analisi forense, questa funzionalità ha un impatto teorico estremamente negativo. Innanzitutto non c'è più alcuna garanzia sulla persistenza dei dati cancellati su disco: una volta che un settore viene segnato come libero, questo potrebbe venire azzerato completamente in qualunque momento. Ma un problema ancora più grave è legato al fatto che il garbage collector è implementato all'interno del controller: questo significa che in teoria potrebbe continuare ad agire anche durante la procedura di acquisizione/analisi dei dati, causando alterazioni irreversibili della prova impossibili da evitare anche con l'uso di un write blocker. In queste condizioni l'uso di sistemi operativi appositi per l'analisi forense o di strumenti come il write blocker, che si frappone tra disco e computer, potrebbe non essere sufficiente a bloccare l'azione del garbage collector.
Questa funzionalità è spesso affiancata, a livello di sistema operativo, dal supporto per un comando ATA chiamato TRIM. Tale comando permette al sistema operativo di comunicare all'SSD quali blocchi contengono dati marcati come cancellati nella tabella di allocazione dei file e possono essere sottoposti a garbage collection. Ad esempio, quando un file viene cancellato, l'OS invia un comando di TRIM relativo sia al file che a tutti i riferimenti ad esso, che viene accodato ed eseguito non appena il disco entra in stato di idle. Come mostrato in [5] anche questa funzionalità ha un effetto negativo sulla recuperabilità dei dati, poiché vi è un forte rischio che i file rimangano fisicamente su disco per non più di pochi secondi dopo la loro cancellazione logica.

2.3.3.2 Errori di I/O dovuti a disturbi

Come già accennato, le memorie flash NAND (soprattutto MLC) sono vulnerabili a cambiamenti del valore logico memorizzato (*bit flip*) a causa di disturbi legati ad attività di lettura o scrittura in celle adiacenti. Esistono due tipi di disturbi.

- Disturbi di lettura, che si verificano quando una cella che non è sottoposta a lettura riceve una tensione elevata. Le celle sottoposte a questo stress sono sempre nel blocco sotto lettura, ma in una pagina che non viene letta. Le probabilità che si verifichi un disturbo di lettura sono molto inferiori rispetto a quelle di un disturbo di scrittura.

- Disturbi di scrittura, che si verificano quando una cella che non è sottoposta a scrittura riceve una tensione elevata. Le celle sottoposte a questo stress sono sempre nel blocco sotto scrittura, e possono essere o nella pagina sotto programmazione o in una qualunque altra pagina all'interno del blocco.

Cancellare una cella causa un reset allo stato originario, eliminando di conseguenza anche gli errori legati a questi disturbi. Per risolvere questo problema, nei controller flash viene spesso implementato un algoritmo di error detection/correction (EDC/ECC) per rilevare e correggere i bit flip prima che i dati arrivino all'host. Gli algoritmi di ECC più diffusi nelle memorie flash NAND sono Reed-Solomon, Hamming e BCH (Bose, Ray-Chaudhuri, Hocquenghem) [6] [7].

2.3.3.3 Conservazione dei dati

Una volta programmato il valore di una cella, è importante che tale valore rimanga valido per un lungo periodo, idealmente finché l'utente non lo vuole modificare. Nelle memorie flash la conservazione dei dati per un periodo accettabile (solitamente definito come 10 anni) può essere garantita mantenendo un livello stabile di tensione sulla cella. Tuttavia perdite di carica attraverso il floating gate (chiamate *charge drift*) possono causare con il passare del tempo lente variazioni del valore di tensione. Tale nuovo livello di tensione può essere erroneamente scambiato per un diverso valore logico. Il tempo di conservazione dei dati è inversamente proporzionale al numero di cicli di scrittura e cancellazione e proporzionale alle dimensioni dei transistor utilizzati: i blocchi che sono stati cancellati molte volte riescono a conservare i dati per un tempo inferiore rispetto a blocchi che hanno subito meno cancellazioni. La capacità di conservazione dei dati delle NAND MLC è di ordini di grandezza inferiore rispetto a quella delle NAND SLC. Per ridurre l'error rate viene solitamente usato un algoritmo di ECC. Bisogna inoltre considerare l'importanza di queste limitazioni nella conservazione dei dati in ambito forense: un tempo di data retention così limitato potrebbe ragionevolmente portare, in tempi paragonabili alla durata di un processo, a cambiamenti casuali in una fonte di prova.

2.3.3.4 Bad block

Se i sistemi di EDC/ECC rilevano un errore non rimediabile l'area va marcata come non valida. Poichè la più piccola area cancellabile è il blocco, per ogni errore irrimediabile rilevato in una pagina l'intero blocco a cui la pagina appartiene va indicato come bad block, così che non vengano più effettuate richieste di accesso al blocco. Esistono due tipi di bad block nelle memorie flash NAND:

- Bad block iniziali: a causa di vincoli di produzione e del tentativo di mantenere bassi i costi di produzione, ogni memoria NAND è prodotta mediamente con un numero di bad block che varia fra il 2% (NAND flash SLC) e il 5% (NAND flash MLC). Il primo blocco fisico è invece sempre garantito come leggibile e senza errori.
- Bad block accumulati: durante l'arco di vita del device, a causa di cicli multipli di lettura/cancellazione gli elettroni intrappolati nel dielettrico possono causare uno shift permanente nei livelli di tensione delle celle. Quando lo shift di tensione supera una certa soglia, questo può essere interpretato come un fallimento di lettura, scrittura o cancellazione e il blocco deve essere marcato come non valido.

Il firmware del controller usa una Bad Block Table per tenere traccia di entrambi i tipi di bad block. Questo non solo contribuisce alle garanzie di integrità dei dati, ma migliora anche le performance eliminando il bisogno di effettuare ripetute operazioni di scrittura dovute a dati mappati ripetutamente sullo stesso bad block. Esistono due diverse strategie per la gestione dei bad block [8]:

- Skip Bad Block (SBB): quando un bad block viene rilevato, il file system passa automaticamente al blocco utilizzabile successivo.
- Reserve Block Area (RBA): Un'area dedicata viene usata come pool di blocchi validi utilizzabili come rimpiazzo dei bad block.

2.3.3.5 Numero limitato di scritture

Le operazioni elettriche necessarie alla scrittura/cancellazione di una cella causano dei danni allo strato di ossido presente nel transistor. Con il ripetersi di tali operazioni il danno aumenta di entità, rendendo sempre più difficile per gli elettroni attraversare questo strato. Una volta che diventa impossibile effettuare l'operazione con le soglie standard di tensione, il blocco va marcato come bad block e sostituito.

Oltre alle tecniche di ECC e bad block management, questo problema può essere prevenuto anche con algoritmi di wear leveling. Il controller mantiene traccia, attraverso l'uso

di una apposita tabella, della mappatura tra l'indirizzo logico delle celle conosciuto dal sistema operativo e il rispettivo indirizzo fisico. Questa mappatura può essere cambiata dinamicamente, con lo scopo di distribuire uniformemente il numero di scritture/cancellazioni tra tutte le celle. Dati frequentemente aggiornati vengono quindi periodicamente spostati in celle meno utilizzate del disco, marcando i settori ospitanti le vecchie versioni del file come spazio libero. Questa è una funzionalità teoricamente vantaggiosa dal punto di vista dell'analisi forense. Se le copie multiple dei file fossero effettivamente visibili dall'esterno e recuperabili, questo renderebbe la cancellazione dei file molto più difficile. Infatti, anche se la copia più recente non fosse recuperabile, il disco potrebbe essere disseminato di residui di versioni precedenti del file pronte ad essere analizzate per estrarre informazioni utili.

Molti produttori, infine, vendono dischi la cui capacità fisica è superiore a quella logica accessibile dal sistema operativo. Questa differenza di capacità è chiamata *over-provisioning pool*. Le celle in eccesso vengono usate come sostitute di quelle accessibili una volta che queste ultime non sono più utilizzabili aumentando così l'affidabilità del dispositivo. La sostituzione viene effettuata in modo trasparente, cambiando la mappatura tra indirizzo logico e fisico.

2.3.4 La struttura logica della memoria

La più piccola area di una memoria flash indirizzabile per un'operazione di scrittura è chiamata pagina, ed è formata da tutte le celle di memoria in una stessa wordline. La dimensione di una pagina è solitamente un multiplo di 512 byte. Ogni pagina è suddivisa in *usable area*, dove i dati sono memorizzati, e *spare area*, usata per i metadati. Il contenuto della spare area non è standardizzato, ma alcuni esempi di metadati sono il numero della pagina, l'erase block di appartenenza, un dirty bit che indica se la pagina è aggiornata o meno e gli EDC/ECC. La dimensione standard della spare area è di 64 o 128 byte.
Analogamente, un blocco è la più piccola area di una memoria flash che può essere azzerata. Un blocco può consistere di 32, 64 o 128 pagine. I blocchi possono essere raggruppati in *erase zone* le cui dimensioni vanno da 256 a 1024 blocchi. Le zone possono essere usate per la gestione dei bad block, conservando entro ogni zona un certo numero di blocchi di riserva da sostituire ad eventuali blocchi corrotti nella stessa zona.

In Figura 2.6 è possibile vedere un esempio di suddivisione di una memoria flash NAND da 8 Gb in pagine e blocchi.

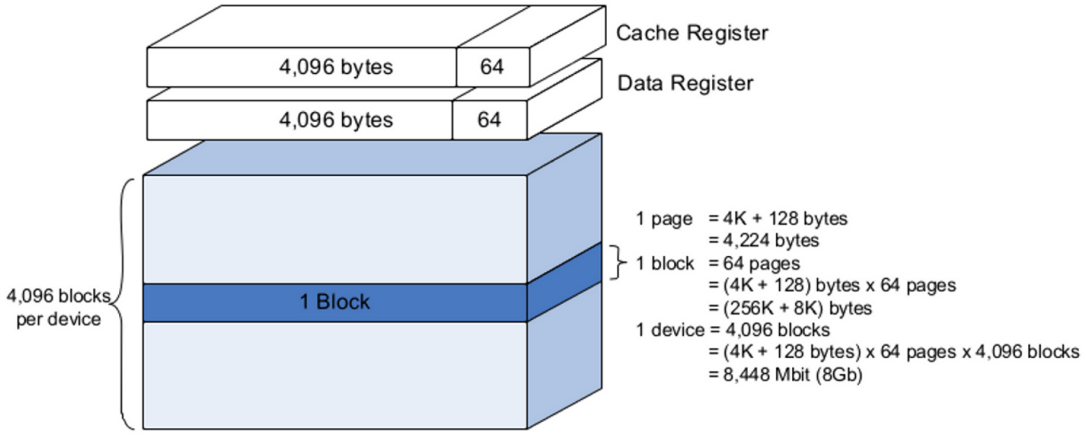


FIGURA 2.6: Una possibile struttura di pagine e blocchi di una memoria flash NAND (8Gb SLC 50nm).

2.3.5 I dischi a stato solido

Le ottime prestazioni di I/O delle memorie NAND hanno rapidamente portato, una volta calati i prezzi di produzione, alla nascita di periferiche di archiviazione completamente basate su questa tecnologia: i Solid State Drive (SSD). Poichè la banda delle singole memorie flash resta limitata, i dischi a stato solido sono costruiti come array di memorie NAND. Le scritture avvengono quindi in parallelo, effettuando striping delle pagine logiche con un approccio simile a quello usato nelle configurazioni RAID-0.
Un BUS seriale collega ogni memoria flash a un controller. Il controller riceve e processa le richieste proventienti dall'host attraverso un'interfaccia di connessione (ad es. SATA), occupandosi di inviare comandi e trasferire dati da (o verso) l'array di memorie flash. La lettura di una pagina avviene prima trasferendo i dati dalla memoria al registro del piano e poi inviandoli attraverso il BUS al controller. La scrittura avviene eseguendo le operazioni in ordine opposto. Alcuni SSD hanno anche un buffer RAM esterno per il caching di dati o metadati.
Ogni memoria flash è suddivisa in *die* (chip), a loro volta suddivisi in più piani. Un piano contiene solitamente migliaia di blocchi (ad es. 2048) e uno o due registi delle dimensioni di una pagina come buffer di I/O. Un blocco contiene di solito da 64 a 128 pagine.

Il controller è il componente centrale dell'SSD. Esso contiene molti elementi importanti, tra cui un microcontrollore, i Flash Interface Module (che si occupano di connettere fisicamente e logicamente il controllore alle singole memorie NAND), un buffer e una memoria dedicata per gli ECC. All'interno del controller è inoltre implementato un componente fondamentale: il Flash Translation Layer (FTL). Il suo ruolo è quello di esporre un array di blocchi logici ai componenti di più alto livello, emulando il comportamento di un classico hard disk rotazionale. L'host può quindi ignorare tutte le complessità legate alle memorie flash, che vengono automaticamente gestite dal controller. Ad esempio il

FTL si occupa di gestire alcuni dei problemi accennati in precedenza, tra cui la gestione dei bad block e il mapping tra indirizzo logico e fisico (legato alla funzionalità di wear leveling e all'over-provisioning) [9]. In Figura 2.7 è possibile vedere un esempio dell'architettura interna di un disco a stato solido e del suo controller.
Per fornire all'utente un livello di sicurezza superiore, alcuni controller implementano una funzionalità di cifratura automatica dei file che utilizza una chiave memorizzata internamente e non modificabile. Questo pone degli ostacoli all'analisi forense del dispositivo simile a quelli già citati per la funzionalità di compressione. Infatti, se si volesse acquisire il contenuto delle memorie senza passare per il livello di astrazione del controller, non si avrebbe alcuna garanzia sulla leggibilità dei dati ottenuti, che potrebbe dipendere dalla conoscenza della chiave e dalla robustezza dell'algoritmo di cifratura.

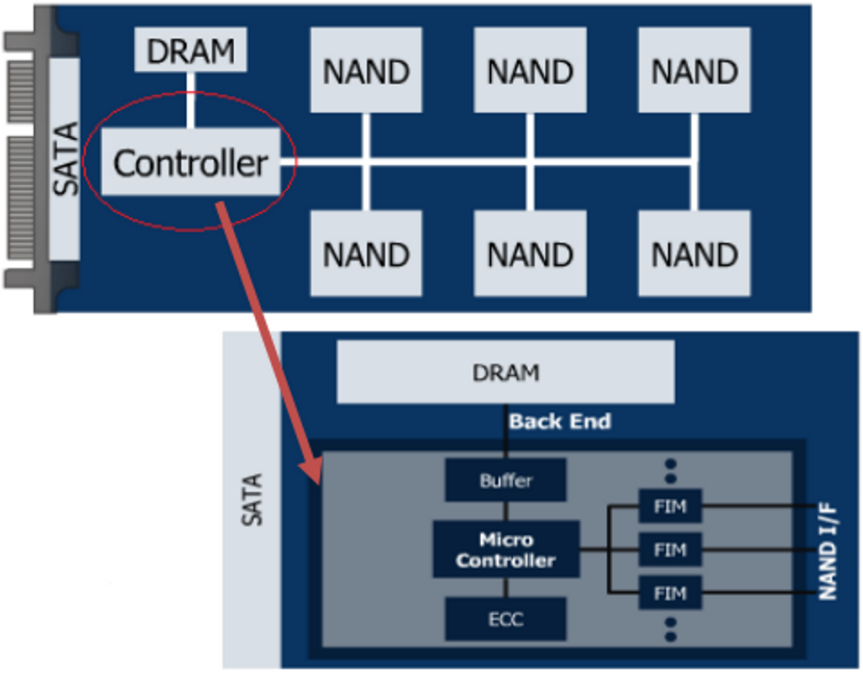


FIGURA 2.7: Esempio di architettura di un disco a stato solido e del suo controllore.

Poiché gli SSD sono una tecnologia piuttosto recente, le tecnologie utilizzate si evolvono velocemente, ed è difficile avere informazioni dettagliate e aggiornate. Ogni produttore cerca di sviluppare innovazioni capaci di rendere i suoi dischi più affidabili e veloci, nascondendone però i dettagli implementativi. Le tecnologie e gli algoritmi implementati, di complessità molto superiore a qualunque disco magnetico, sono infatti spesso del tutto mascherati dal livello di astrazione garantito dal FTL.

2.3.6 Un confronto fra SSD e HDD

La prima differenza è l'assenza di parti mobili all'interno degli SSD. Gli hard disk tradizionali, infatti, contengono molti componenti in movimento durante le operazioni di lettura e scrittura. Ciò, oltre a introdurre vari tipi di latenza dovuti a limiti fisici dei componenti in movimento, rende gli HDD vulnerabili a vibrazioni ed urti. I dischi a stato solido, grazie all'assenza di componenti in movimento, garantiscono prestazioni e resistenza superiori a fronte di consumi decisamente inferiori. Inoltre, poichè gli HDD sono spesso il vertice dei livelli di memorizzazione (cioè il più capiente e meno prestazionale), essi costituiscono il collo di bottiglia delle prestazioni dell'intero sistema, rimuovendo il quale le performance dei sistemi su cui sono montati aumentano sensibilmente. Nonostante l'evidente superiorità dei dischi a stato solido, la complessità intrinseca di questa tecnologia ed il conseguente costo per GB elevato ha finora limitato la diffusione su larga scala.

	2.5" SATA 3.8Gbps HDD	**2.5" SATA 3.8Gbps SSD**
Meccanismo	Piatti magnetici rotanti	NAND Flash
Densità	80 GB	64 GB
Peso	365 g	73 g
Performance	Lettura: 59 MB/s	Lettura: 100 MB/s
	Scrittura: 60 MB/s	Scrittura: 80 MB/s
Tempo di Seek	Variabile ($\geq$ 10 ms)	Costante (0.1 ms)
Consumo di Potenza	3.86 W	1 W
Durabilità	MTBF $<$ 0.7 Mhours	MTBF $>$ 2Mhours

Tabella 2.2: Differenze principali tra HDD e SSD

2.4 Stato dell'arte

I dischi a stato solido sono una tecnologia recente, esistono quindi pochi lavori di ricerca orientati al loro studio. La possibilità di approcciare lo studio di questi dischi in modi differenti inoltre, ha comportato la presenza di risultati contrastanti a seconda del punto di vista intrapreso nella ricerca.

Graeme B. Bell e Richard Boddington [10] si sono concentrati su un'analisi di tipo black-box, verificando la possibilità che, durante acquisizioni forensi e nonostante precauzioni come l'uso di sistemi operativi forensi o write blocker, i complessi controller dei dischi a stato solido possano avviare meccanismi di garbage collection che portino all'azzeramento involontario e incontrollabile di porzioni del disco. Il lavoro ha ottenuto risultati preoccupanti dal punto di vista nell'analisi forense, poichè questi meccanismi di azzeramento

agiscono in tempi sufficientemente rapidi da rendere difficile identificare e rimediare al problema prima che la loro azione sia conclusa. Sebbene realistico, lo scenario proposto da Bell e Boddington riguarda tuttavia un caso piuttosto improbabile nella realtà, cioè un disco sottoposto ad una operazione di formattazione rapida immediatamente precedente ad una analisi forense. In ogni altro caso però, sia essa una formattazione completa (con azzeramento dei dati) o l'analisi forense di un disco che non abbia subito alcuna operazione di formattazione, i risultati ottenuti sarebbero analoghi a quelli di un disco magnetico. Inoltre il numero molto limitato di esperimenti effettuati, riguardanti il caso limite di un disco completamente riempito, potrebbero non aver indagato a sufficienza e in modo completo il comportamento del controller, che potrebbe essere influenzato vari parametri non presi in considerazione, tra cui ad esempio la percentuale di riempimento del disco o il tipo di file system utilizzato.

Un esperimento simile a quello di Bell e Boddington è stato poi riprodotto su molti più dischi e sistemi operativi da Christopher King e Timothy Vidas [5]. In particolare i test sono stati eseguiti su Windows XP, Windows 7 e Ubuntu 9.04, installati sia su dispositivi dotati di supporto TRIM che su dischi privi di tale funzionalità, utilizzando file di diverse dimensioni e simulando diverse condizioni d'uso. Come impostazioni di formattazione sono state utilizzate quelle predefinite fornite da ognuno dei sistemi operativi. I risultati hanno mostrato che laddove sia presente il supporto al comando TRIM, l'invio di tale comando (da parte del sistema operativo o manualmente) rende impossibile recuperare i dati cancellati. In assenza di tale supporto invece è possibile una recuperabilità almeno parziale dei dati. Dall'esito degli esperimenti la percentuale di dati recuperabili sembra dipendere dalla politica di formattazione dei singoli sistemi operativi e dalla dimensione dei file utilizzati.

Prima di loro anche Antonellis [11] aveva testato la recuperabilità dei file cancellati dai dischi a stato solido, in particolare verificando gli effetti del comando standard di cancellazione utilizzato da Windows. É importante sottolineare che il lavoro è stato pubblicato nel 2008, anno in cui nessun sistema operativo disponeva di supporto TRIM, quindi qualunque perdita di dati rilevata non sarebbe potuta dipendere da tale funzionalità. Al termine degli esperimenti tutti gli strumenti di analisi forense utilizzati sembravano in primo luogo riuscire a recuperare i dati, ma un'analisi del contenuto esadecimale dei file recuperati mostrava solamente una sequenza di 00. Le uniche informazioni che era stato possibile recuperare consistevano quindi nei metadati dei file cancellati.

Wei, Gupp, Spada e Swanson [12] hanno invece adottato un approccio di tipo white box, che consiste in una lettura diretta dei chip di memoria senza passare attraverso il controller. Lo scopo del loro lavoro, in particolare, consiste nel verificare l'efficacia di diversi sistemi di cancellazione sicura sia di interi dischi che di singoli file. Per aggirare le astrazioni ed i mascheramenti compiuti dai controller è stata sviluppata una piattaforma hardware basata su FPGA [13] capace di accedere direttamente ai chip NAND. Sono

stati testati molti protocolli di cancellazione dei file, ma nessuna tecnica software è stata in grado di rimuovere completamente i dati a causa della presenza di copie multiple dei file. La ragione della presenza di copie multiple dei file e della impossibilità di effettuare un completo azzeramento è interpretata come effetto collaterale di tecnologie come il wear leveling, utilizzate per garantire una vita più lunga ai dispositivi SSD. Wei giunge quindi alla conclusione che, utilizzando un approccio di questo tipo, i dischi a stato solido offrano possibilità di recupero dei dati anche superiori a quelle offerte da un disco tradizionale, poichè gli stessi meccanismi che rendono affidabile il dispositivo rendono di fatto quasi impossibile il totale azzeramento dei dati in esso contenuti.

Al di fuori degli SSD esistono tuttavia molti lavori di ricerca orientati all'analisi forense di altri dispositivi basati su memorie flash. Ad esempio in Breeuwsma [14] vengono proposte diverse metodologie di basso livello per l'acquisizione di una copia completa della memoria in memory stick USB e dispositivi mobili, con uno sguardo anche all'analisi dei file system. In particolare vengono presentati tre approcci per l'acquisizione dei contenuti di memorie flash facenti parte di sistemi embedded: l'uso di software dedicati (flasher tools), un metodo basato su JTAG e il dissaldamento delle memorie per la lettura con strumenti esterni. Il lavoro si occupa inoltre di spiegare come i dati ottenuti possono essere tradotti per renderli comprensibili ai più comuni software di analisi forense (ad esempio Encase o TSK).
Billard e Hauri [15] si occupano invece di proporre una metodologia di carving alternativa ottimizzata per analizzare dump di memorie flash. Gli algoritmi comunemente utilizzati hanno infatti diverse limitazioni, tra cui il fatto che si basano sulla comune suddivisione in cluster effettuata dai file system, diversa dalla suddivisione in blocchi propria dei dispositivi a stato solido. La traduzione da blocco a cluster viene comunemente eseguita dal FTL, le cui informazioni non sono tuttavia disponibili nel dump da analizzare. Il lavoro si occupa quindi di presentare il modello formale dietro a un algoritmo di carving indipendente dalla struttura del file system, che basandosi su una conoscenza a priori parziale dei file presenti su disco decompone il dump in componenti elementari, classifica ogni parte in base al formato del file a cui dovrebbe appartenere e infine riordina tali componenti per ricostruire i file.
Completamente orientato ai memory stick è invece Phillips [16], che si occupa del recupero di dati da dispositivi danneggiati o cancellati elettronicamente. Dopo aver riempito completamente di file testuali e audio dei dispositivi formattati con il file system FAT-16, gli autori hanno proceduto a danneggiarli intenzionalmente attraverso metodi che possono essere utilizzati quando si vogliono distruggere frettolosamente dei memory stick: sovra-voltaggio con una batteria da automobile a 12V, immersione in acqua, incenerimento in un contenitore pieno di benzina, calpestamento, martellamento, cottura in un forno a microonde e infine un proiettile sparato a distanza di 5 metri. Le memorie flash

sono state quindi rimosse e lette direttamente con un microcontrollore. In quattro casi su sette (sovra-voltaggio, immersione, benzina in fiamme e calpestamento) i dati sono risultati comunque recuperabili.
Sempre applicato ai memory stick, ma estendibile anche ad altri dispositivi basati su memorie flash, è il lavoro di Templeman e Kapadia [17], che si occupa di studiare e implementare un attacco per accelerare l'usura delle celle delle flash. L'applicazione agisce cambiando quanti più bit possibile nel corso di ogni operazione di scrittura, con lo scopo di massimizzare la write amplification. Il contenuto delle scritture è casuale, così come la dimensione dei file e il duty cycle, e ogni scrittura è immediatamente seguita da una cancellazione. L'applicazione, inoltre, è implementata in modo da agire con una politica che la rende non rilevabile attraverso l'osservazione delle performance del sistema a cui il dispositivo sotto attacco è collegato. I test svolti nel corso del lavoro hanno dimostrato che un dispositivo che in condizioni di uso normale può resistere ad elevati carichi di scritture quotidiane per più di cinque mesi viene usurato dall'attacco in un tempo che varia dai 14 ai 29 giorni. Il documento si conclude proponendo alcune possibili difese a questo attacco, implementabili a livello di sistema operativo o di controller.
Un lavoro interamente dedicato alla telefonia mobile si può trovare nel lavoro di Luck e Stokes [18], concentrato sul recupero di file video da dispositivi con memorie di tipo NAND. Ricostruendo innanzitutto la partizione FAT, di cui viene fornita nel lavoro una dettagliata descrizione, vengono recuperate tutte le liste di cluster utilizzate dai vari file. L'articolo si concentra poi sull'uso dei metadati dei formati MPEG-4 e 3GP per estrarre contenuti video, con particolare attenzione al recupero degli I-Frame, che laddove presenti garantiscono che il video recuperato sia almeno in parte ancora visibile.
Infine in Skorobogatov [19] viene presentato un approccio più vicino all'elettronica per la ricerca di tracce di dati cancellati in memorie flash di tipo NOR, dimostrando che è in alcuni chip possibile trovare residui di carica all'interno dei floating gate anche dopo decine di cicli di cancellazione. Ciò permette di distinguere anche in seguito ad un azzeramento quali celle sono state precedentemente programmate e quali no. Gli esperimenti sono stati effettuati sia con metodi non invasivi (esposizione a raggi UV) sia con metodi semi-invasivi (ionizzazione attraverso un fascio laser).

3. Una metodologia per guidare l'analisi di dischi a stato solido

Questo capitolo ha lo scopo di illustrare nel dettaglio la procedura di analisi dei dischi che abbiamo sviluppato. I singoli test sono basati ognuno su una specifica funzionalità del disco e per ognuno di essi vengono illustrate tutte le scelte che ci hanno portato alla definizione dei vari passi. Da affiancare agli esperimenti proponiamo anche una metrica di classificazione dei dischi che indichi il loro livello di analizzabilità forense, basata sui risultati dei test proposti.

3.1 Concetti preliminari

La presenza di un'ampia varietà di produttori e di controller, non tutti direttamente testabili nel corso del nostro lavoro, ci ha spinto a realizzare una metodologia di testing ed una metrica di classificazione che non fossero orientate al singolo disco o controller, ma piuttosto alle funzionalità implementate. Per evitare che i risultati fossero influenzati da più fattori indistinguibili tra loro, abbiamo deciso di studiare metodologie di analisi distinte per ogni caratteristica dei dischi a stato solido rilevante dal punto di vista dell'analisi forense. Alcune di queste procedure hanno diversi passi in comune, quindi nulla impedisce ad un lettore che voglia riprodurre i test di fondere alcuni dei protocolli proposti, ove sia possibile. In appendice è possibile trovare tutti i tool sviluppati per l'esecuzione e l'automatizzazione dei passi del test.

Prima di presentare i protocolli può essere utile definire il significato di alcuni termini e procedure ricorrenti in vari test:

- Quando parliamo di *sistema operativo ad uso forense*, la caratteristica che più ci interessa è che il sistema operativo non monti automaticamente i dischi collegati al computer e non effettui scritture in background su di essi. Ciò serve per garantire che eventuali modifiche del contenuto del disco siano legate esclusivamente

ad azioni del controller o volontarie, e non all'azione del sistema operativo. Non è indispensabile ma preferibile utilizzare un OS sviluppato appositamente per l'analisi forense, poichè sebbene sia possibile disabilitare manualmente l'automount ed alcune delle operazioni di background in ogni generica distribuzione Linux, pensiamo che l'uso di OS dedicati allo scopo sia la via più semplice e fornisca maggiori garanzie.

- Nei protocolli di test saranno spesso presenti due tipi distinti di formattazione del disco. La prima, la formattazione totale, è intesa come una formattazione che causi l'azzeramento di ogni settore del disco. Utilizziamo questa operazione per riportare il disco ad uno stato pulito, evitando che l'esito dell'esperimento corrente sia influenzato da residui degli esperimenti precedenti. L'altro tipo di formattazione, definita rquick format, è intesa come una formattazione che riscriva soltanto la tabella di allocazione dei file, senza un loro reale azzeramento sul disco.

- Quando parliamo di disco in stato di *idle* intendiamo un disco alimentato ma su cui non sono in esecuzione operazioni di I/O. Analogamente, un disco *attivo* è un disco alimentato su cui sono in esecuzione operazioni di I/O.

- Per *calcolo dell'hash* intendiamo l'utilizzo sia dell'algoritmo MD5 che SHA-1 perché, per quanto sia bassa la probabilità di collisioni (ovvero che esistano due file distinti con il medesimo hash), la probabilità di un conflitto simultaneo su più algoritmi di hash è estremamente bassa rendendo il confronto degli hash una operazione ancora più affidabile rispetto all'utilizzo di un unico algoritmo.

3.2 Metriche di analizzabilità forense

Per classificare i dischi al termine dei vari test abbiamo definito tre distinte metriche di valutazione. Il problema è infatti composto da varie dimensioni che non possono essere aggregate senza causare una perdita di informazioni. Sono state quindi definite le seguenti classificazioni: una metrica riguardante l'analizzabilità attraverso metodi black box, una seconda legata all'analizzabilità attraverso metodi white box e un'ultima legata alla capacità del dispositivo di garantire l'integrità dei dati nel corso dell'analisi.

Le classificazione dei dischi secondo metriche di analizzabilità black box è la seguente:

- A: Recuperabilità pari a HDD

- B: Recuperabilità parziale

- C: Azzeramento totale

Tale metrica va utilizzata per misurare il comportamento del disco in due situazioni distinte: la recuperabilità dei dati in seguito a un quick format e la recuperabilità di singoli file cancellati attraverso il sistema operativo. Infatti, come vedremo nel Capitolo 4, dai nostri esperimenti è emerso che il comportamento di uno stesso disco, nelle medesime condizioni di test, in seguito a queste due operazioni può risultare differente. A proposito delle condizioni di test, è importante sottolineare che la classificazione appena riportata non è legata solo ed esclusivamente al disco, ma ad una triade <disco, file system, OS> poichè ognuna di queste componenti, interagendo con le altre, impatta in modo differente sulla recuperabilità dei dati.

Riguardo all'analizzabilità white box, invece, le classi possibili sono le seguenti:

- 1: Recuperabilità dati non visibili da analisi black box
- 2: Recuperabilità pari a black box
- 3: Presenza di cifratura/compressione effettuata a livello di controller
- 4: Impossibilità di accesso a memorie per limiti fisici

Quest'ultima classe deriva dal fatto che alcuni produttori utilizzano memorie NAND proprietarie senza una piedinatura accessibile a cui collegare gli strumenti di analisi.

Infine, per quanto riguarda l'integrità dei dati su disco, la classificazione è binaria:

- $+$: Hash stabile
- $-$: Hash instabile

Come già accennato, infatti, in molte legislazioni (ad esempio quella americana) l'hash ha un valore legale fondamentale poichè garantisce che il contenuto del disco non sia stato alterato in seguito alla fase di acquisizione della prova. Se a causa di alcune funzionalità (ad es. garbage collection) l'hash dovesse cambiare durante le procedure di acquisizione del disco, potrebbe diventare difficile dimostrare che i dati su disco non siano stati modificati in fase di analisi.

3.3 Protocolli black box

Con esperimento black box intendiamo un esperimento effettuato collegando il disco a una porta SATA e osservandone il comportamento secondo i dati visibili attraverso il sistema operativo. Ciò significa che i risultati non rappresentano necessariamente lo stato interno del disco, ma piuttosto ciò che il controller permette di osservare attraverso il suo livello di astrazione. É quindi importante tenere conto della possibilità che il vero comportamento del disco venga mascherato dal controller e quindi non sia rilevabile attraverso questi test.

3.3.1 Stabilità dell'hash

Lo scopo di questo test è verificare la presenza di un ipotetico garbage collector implementato a livello di controller, in particolare per verificare se la sua attivazione possa portare ad alterazioni dell'hash del disco. Poiché i produttori dei controller non forniscono informazioni pubbliche sull'implementazione degli algoritmi di garbage collection possiamo fare due ipotesi riguardo al loro funzionamento. La prima è che agiscano a livello dello spazio di indirizzamento logico, in modo analogo a un comando di TRIM ma senza la necessità di un intervento del sistema operativo; ciò è coerente con i risultati di Bell-Boddington [10] ed è il caso che il nostro protocollo si occupa di analizzare. La seconda è che gli algoritmi ottimizzino il disco in modo trasparente eseguendo operazioni non visibili al di sopra del FTL; questa possibilità andrebbe invece esplorata con un test di tipo white box.

Il diagramma in Figura 3.1 mostra la sequenza di operazioni che abbiamo scelto di eseguire per testare il garbage collector, oltre a una legenda relativa a tutti i diagrammi dei protocolli di test. L'utilizzo del sistema operativo ad uso forense è indispensabile sia per evitare che l'OS esegua scritture di background sul disco, sia per evitare che vi siano azzeramenti dovuti all'effetto di comandi di TRIM. É consigliabile effettuare il monitoraggio in tempo reale durante l'acquisizione degli hash per assicurarsi che non vi siano variazioni nella quantità di settori azzerati su disco nel corso dell'operazione.
La scelta di un periodo di idle di 16 ore ci è sembrata la massima attesa ragionevole perché qualunque algoritmo di ottimizzazione o di garbage collection potesse attivarsi e completare la sua esecuzione. Tuttavia con grande probabilità un intervallo di tempo di tale lunghezza non è necessario, poiché in [10] Bell e Boddington rilevano risultati significativi in intervalli di tempo che variano fra i 3 e i 30 minuti. Abbiamo inoltre deciso effettuare il test con riempimenti del disco a diverse percentuali, supponendo che alcuni produttori possano scegliere di implementare politiche di garbage collection dipendenti dallo spazio disponibile sul disco.

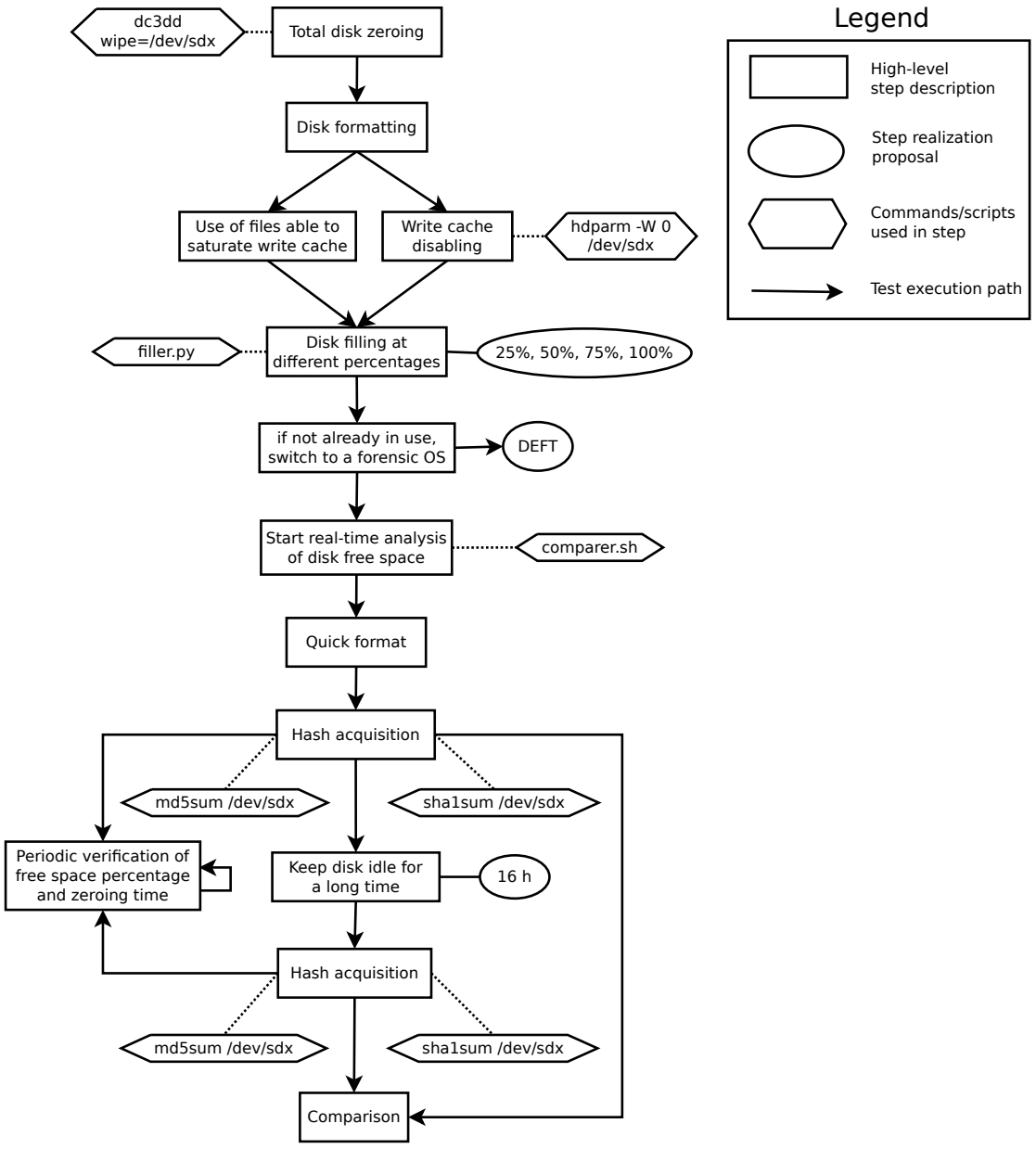


FIGURA 3.1: Protocollo di analisi della stabilità dell'hash

Interpretazione dei Risultati

Il confronto degli hash può avere soltanto due esiti, ognuno corrispondente ad una classe di integrità dei dati:

- Gli hash sono uguali: classe $+$
- Gli hash sono diversi: classe $-$

3.3.2 Aggressività garbage collection

Così come il precedente, questo test si occupa di studiare gli effetti di un ipotetico garbage collector implementato a livello di controller. Lo scopo del test è tuttavia differente, poichè in questo caso consiste nel misurare l'aggressività della politica di azzeramento dei dati. Il contenuto dei file utilizzati non è importante, ma la loro dimensione sì, soprattutto se non si è sicuri del funzionamento del comando di disabilitazione della write cache, che potrebbe venire mascherato dal controller. Nel caso in cui la write cache sia attiva, infatti, vi è il rischio che piccoli file non vengano scritti fisicamente su disco.

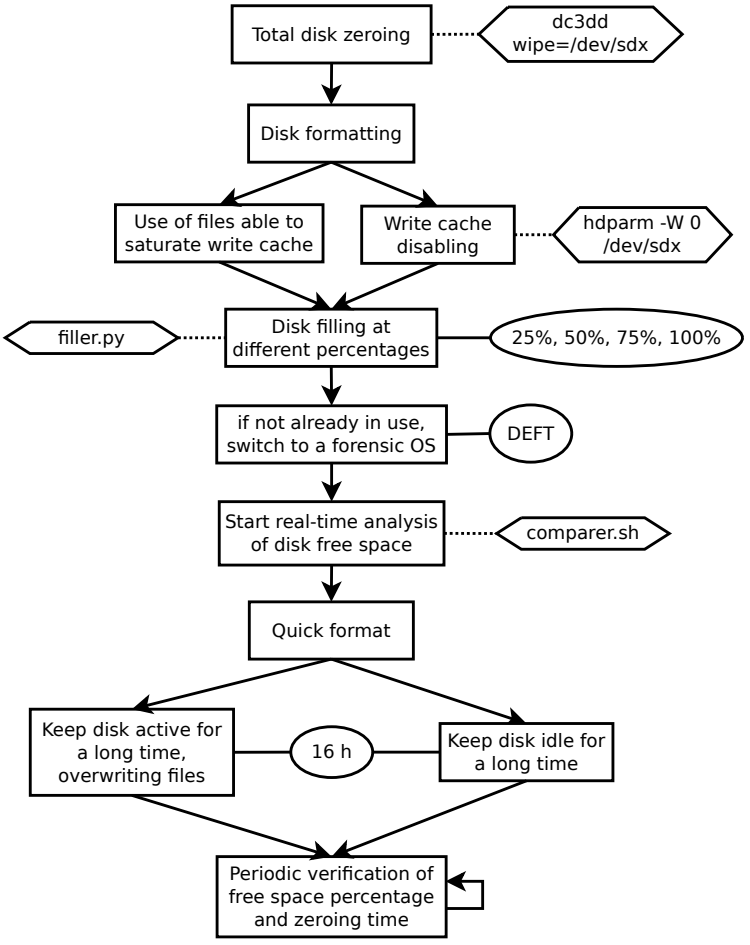


FIGURA 3.2: Protocollo di analisi dell'aggressività del garbage collector

Poiché non esistono informazioni pubbliche dettagliate sulle condizioni di attivazione dei garbage collector implementati all'interno dei controller dei dischi a stato solido, abbiamo voluto testare tutti i casi teoricamente possibili. Uno dei rami di analisi verifica il comportamento del disco in fase di idle, mentre nell'altro il disco viene mantenuto attivo attraverso ripetute sovrascritture di un singolo file. In ambo i casi il test viene ripetuto a diverse percentuali di riempimento. Per questo test valgono tutte le considerazioni già riportate per gli esperimenti sulla stabilità dell'hash.

Interpretazione dei Risultati

Ai fini della classificazione questo test, da solo, non è sempre sufficiente a decidere la classe di analizzabilità black box. Tuttavia può fornire delle indicazioni sulla scelta che vanno poi integrate con gli esiti di altri test:

- Se non si attiva nessun garbage collector, la classe potrebbe essere A
- Se il garbage collector azzera solo parte dei dati, la classe potrebbe essere B
- Se il garbage collector azzera completamente i dati, la classe è C

Abbiamo usato una forma ipotetica nelle prime due classi poichè questo test è strettamente legato al successivo test di verifica delle funzionalità di TRIM, e quindi per la classificazione abbiamo scelto di considerare il comportamento più aggressivo tra i due test, eventualmente confermandolo con il test di recuperabilità. É tuttavia presente anche un altro aspetto interessante che esula dalla classificabilità, ovvero l'aspetto temporale. La classificazione si riferisce allo stato del disco dopo l'attivazione della funzionalità di garbage collection, ma se i tempi di attivazione non sono estremamente brevi sarebbe teoricamente possibile prevenirne gli effetti rimuovendo prontamente la fonte di alimentazione del disco.

3.3.3 TRIM (black box)

Questo test è stato studiato con l'obiettivo di verificare come le politiche di TRIM siano implementate nei sistemi operativi che supportano tale comando. Attualmente il comando ha un supporto molto limitato: per quanto riguarda NTFS è supportato e abilitato di default nei soli sistemi operativi Windows 7 e 8, mentre per il filesystem ext4 il supporto è presente solo su distribuzioni Linux basate su kernel $\geq$ 2.6.28 e va abilitato manualmente dall'utente. Come mostrato in Figura 3.3 le direzioni di indagine sono due: la prima consiste nella verifica del tempo di azzeramento e della percentuale di dati azzerati in seguito a un quick format, mentre la seconda consiste nell'utilizzare su singoli file il comando di cancellazione implementato nel sistema operativo e misurare il tempo necessario al loro azzeramento. Poichè alcuni dischi potrebbero applicare politiche di TRIM influenzate dalla quantità di dati presenti su disco, anche in questo caso il test viene ripetuto a diverse percentuali di riempimento. L'azione del TRIM è rilevabile in un lasso di tempo molto breve, quindi è consigliabile eseguire durante il test un'analisi in tempo reale dello spazio libero su disco.

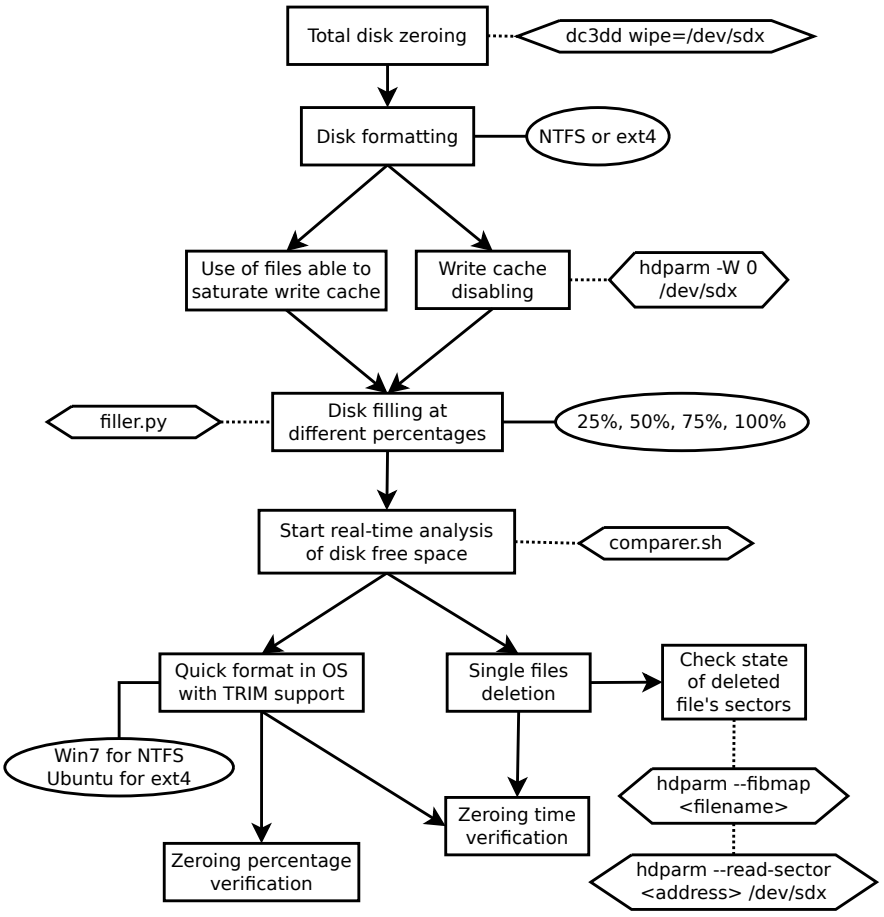


FIGURA 3.3: Protocollo di analisi black box della funzionalità di TRIM

Interpretazione dei Risultati

L'esito del test può fornire informazioni utili sulla classe di analizzabilità black box, sia nel caso di quick format che nel caso del comando di cancellazione del sistema operativo.

- Se non viene rilevata alcuna variazione dello spazio libero su disco, la classe potrebbe essere A
- Se i dati vengono azzerati solo parzialmente, la classe potrebbe essere B
- Se il TRIM azzera completamente i dati, la classe è C

Anche in questo caso, le indicazioni per le prime due classi sono date in forma ipotetica poichè per la classificazione va scelto il comportamento più aggressivo tra quelli rilevati con questo test e con il test sul garbage collector. I risultati vanno inoltre confermati con il test di recuperabilità dei dati.

3.3.4 Recuperabilità dei file

Questo test si propone di valutare, ove gli effetti di TRIM e garbage collection non abbiano portato ad un azzeramento completo, se i dati rimanenti su disco corrispondano poi a file effettivamente recuperabili. A seconda della funzionalità da analizzare, il quick format va effettuato attraverso un sistema operativo rispettivamente con o senza supporto TRIM. Il test effettua un controllo dell'hash su ogni file con l'obiettivo di verificare l'assoluta integrità dei file recuperati. Ciò non toglie che il contenuto di file non integri possa essere comunque comprensibile almeno in parte (ad esempio nel caso in cui si tratti di frammenti di file di testo).

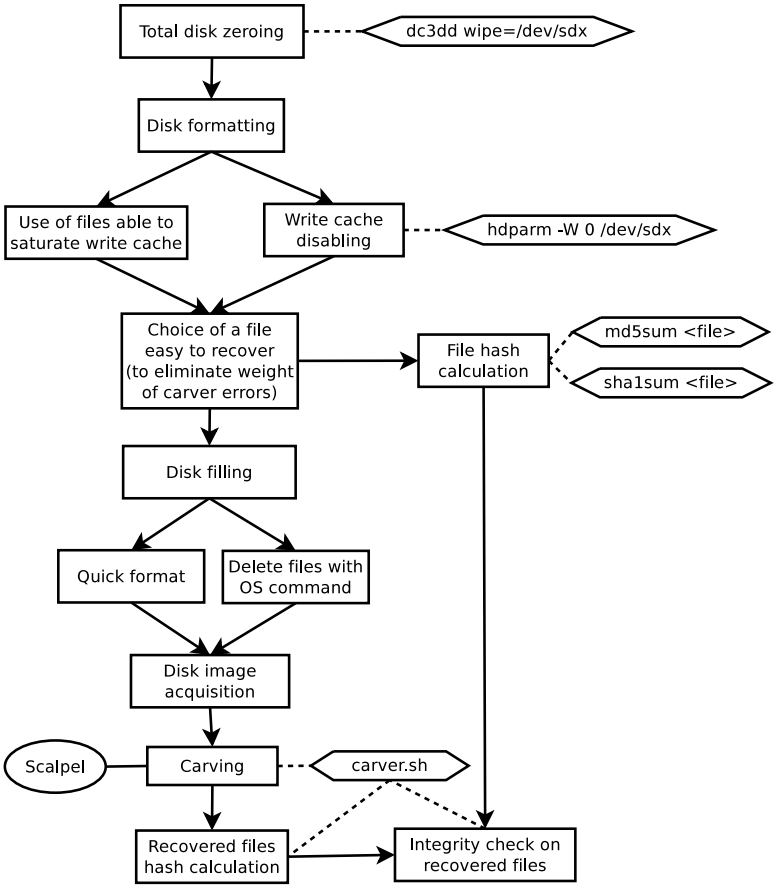


FIGURA 3.4: Protocollo di analisi della recuperabilità di file cancellati

La scelta di un carver specifico, in questo caso *scalpel*, è legata al suo alto livello di accuratezza e di prestazioni nel recupero delle immagini di test utilizzate. Sebbene tutti i carver disponibili funzionino secondo gli stessi principi di base, essi utilizzano algoritmi molto diversi fra loro. É quindi possibile che per altre tipologie di file, o in situazioni di diversa frammentazione, altri carver possano avere prestazioni superiori. Dunque è solo effettuando esperimenti pratici che si può verificare quale sia la scelta migliore per ogni singolo caso.

Come metrica per la valutazione dell'esito del test è stata scelta la percentuale di dati recuperati integri rispetto alla quantità totale di dati presenti su disco. Questa percentuale può essere calcolata in vari modi, ad esempio basandosi sul numero di file recuperati, oppure sul numero di GB. Crediamo tuttavia che non possa essere definito un approccio univoco, poiché l'efficacia di ognuno di essi cambia in base alle condizioni di test. Se usassimo una percentuale basata sul numero di file, ad esempio, la dimensione dei file usati per il test avrebbe un grande impatto a causa del rapporto inversamente proporzionale fra la dimensione del file e la probabilità di recuperarlo, in quanto insieme alla dimensione del file cresce il numero medio di frammenti in cui il file è suddiviso.

Interpretazione dei Risultati

A causa dell'influenza mai completamente eliminabile delle non idealità del carver, abbiamo deciso di definire una soglia di tolleranza per la definizione di recuperabilità totale. Le classi di recuperabilità black box da associare ai risultati del test, sia per il caso di quick format che per l'applicazione del comando di cancellazione del sistema operativo, sono quindi le seguenti:

- Se è sempre possibile recuperare come minimo il 95% dei dati in condizioni di integrità, la classe è A.
- Se è possibile recuperare come integri meno del 95% dei dati, allora la classe è B.
- Se non è mai possibile recuperare nessun dato integro, la classe è C

3.3.5 Compressione

Alcuni dei moderni controller per dischi a stato solido implementano funzionalità di compressione hardware dei dati, per ridurre il fenomeno della write amplification ed aumentare così la vita utile e le prestazioni del device. Se un controller implementa funzionalità di compressione, il rischio è che i dati siano indecifrabili una volta recuperati con metodi white box, quindi è importante verificare la presenza o meno di tale funzionalità.

Il nostro approccio consiste nel confrontare i tempi di scrittura di due blocchi di dati delle medesime dimensioni, uno a bassa entropia (quindi molto comprimibile) e l'altro ad alta entropia (pressoché incomprimibile). Prima della scrittura i blocchi di dati vengono inoltre copiati in memoria primaria, per evitare che la lettura da disco funga da collo di bottiglia. Inoltre, nonostante la nostra procedura invii esplicitamente al disco un comando di disabilitazione della write cache, abbiamo deciso di utilizzare blocchi di dati di dimensioni molto maggiori di tale cache per prevenire il caso in cui il controller mascheri tale comando.

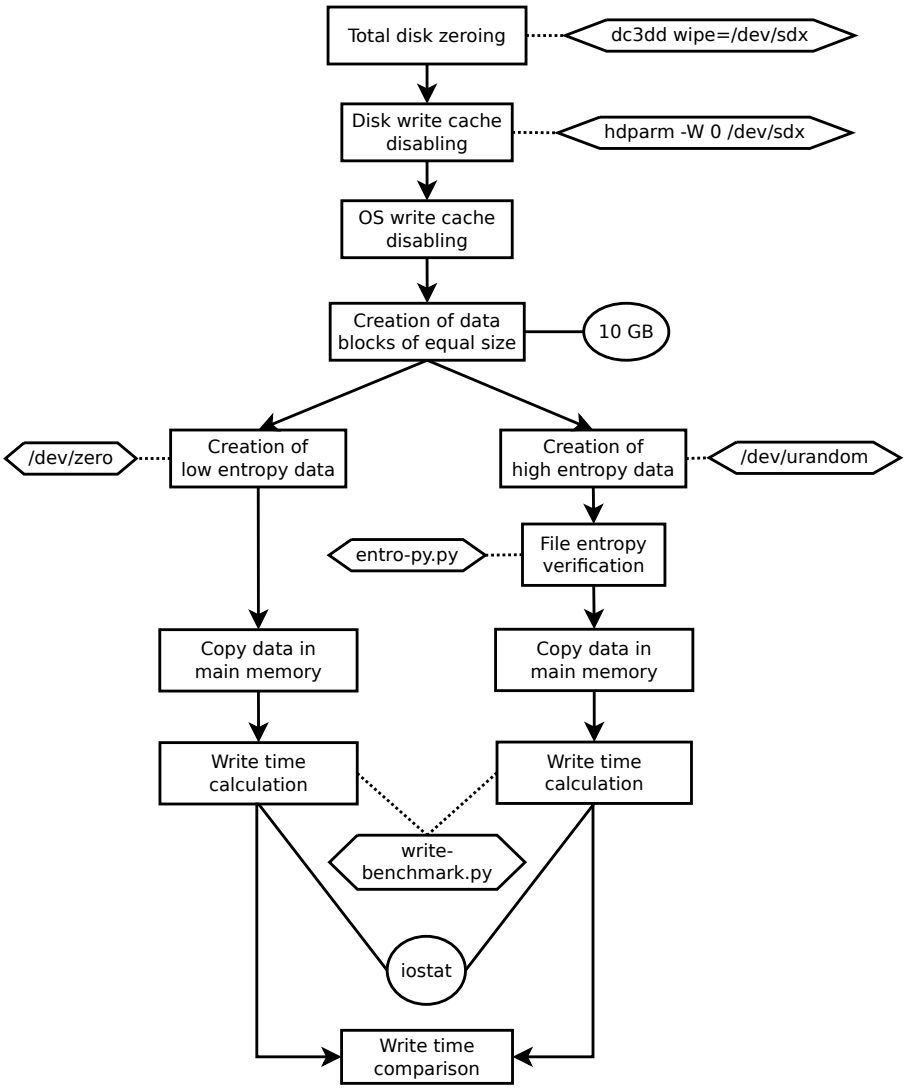


FIGURA 3.5: Protocollo di analisi della funzionalità di compressione

Per generare i dati ad alta entropia ci siamo affidati ad un generatore di numeri casuali non bloccante, dunque l'effettiva entropia dei dati generati è impredicibile ed è necessario verificarla. Il nostro script di verifica si basa sul concetto dell'entropia di Shannon [20], che misura il valore atteso dell'informazione contenuta all'interno di un messaggio, interpretato come una sequenza di variabili casuali indipendenti e identicamente distribuite. In termini informatici tale valore consiste nel numero medio di bit da utilizzare per codificare nel modo più breve (ovvero compresso) possibile un messaggio senza perdite di informazioni. Un messaggio contenente un solo valore ripetuto dall'inizio alla fine avrà quindi il minimo valore di entropia, poichè il messaggio deve contenere solo le informazioni sul singolo valore utilizzato e sul numero di volte per cui va ripetuto; al contrario un messaggio contenente una sequenza di valori non predicibile, come può essere una sequenza casuale di caratteri, avrà entropia massima poichè per non perdere informazione andrà codificato nella sua interezza.

Lo script normalizza inoltre il valore dell'entropia di Shannon rispetto alla dimensione dei dati, così da ottenere un output compreso tra 0 e 1. In questo lavoro consideriamo accettabile come file ad alta entropia un file con entropia di Shannon maggiore di 0.9. Sebbene /dev/random fornisca output con valori di entropia più elevati, per i nostri esperimenti abbiamo scelto di utilizzare /dev/urandom poiché comporta una perdita insignificante per le nostre esigenze e garantisce prestazioni migliori.

All'interno dei dischi a stato solido le funzionalità di compressione (laddove presenti) sono di solito molto efficaci. Fornendo al disco un file molto comprimibile, il numero di scritture da effettuare viene estremamente ridotto senza che l'operazione di compressione introduca overhead, poichè il tempo necessario per eseguirla è inferiore al tempo di accesso alle memorie NAND. Molto visibile è invece la differenza rispetto alla scrittura di un file non comprimibile, soprattutto effettuando l'operazione su una quantità di dati sufficientemente grande da eliminare il peso di eventuali ritardi dovuti al sistema operativo o a momentanei incrementi di prestazioni dovuti all'intervento di una write cache.

Poichè il tempo di sovrascrittura su una cella piena è superiore al tempo di scrittura in una cella vuota è inoltre importante effettuare un azzeramento totale del disco prima del test, per garantire che le condizioni di scrittura siano uniformi. La formattazione non è invece presente nel protocollo in quanto superflua: la funzionalità di compressione è indipendente dal file system, quindi anche operando su dati raw come fa il nostro script i risultati sono ugualmente visibili.

Interpretazione dei Risultati

L'esito di questo test ha le seguenti influenze sulla classificazione del disco riguardo alla recuperabilità white box:

- Se viene rilevato un tempo di scrittura, per file ad alta entropia, significativamente più grande rispetto a file di dimensione equivalente ma a bassa entropia, il controller implementa compressione hardware e la classe di appartenenza è la 3
- Altrimenti è necessario effettuare test differenti per classificare il disco (si veda la Sezione 3.4)

3.3.6 Wear leveling (black box)

L'obiettivo di questo test non consiste nell'analizzare se la funzionalità di wear leveling è presente nei dischi, poiché ci aspettiamo che agisca nel modo più trasparente possibile, senza necessariamente risultare visibile attraverso il controller. Piuttosto il nostro scopo

è studiare quanto trasparente sia tale procedura, cercando di scoprire se sia possibile rilevare tracce della sua attivazione attraverso un'analisi black box, se non qualitativamente almeno quantificandone gli effetti. Qui più che altrove è importante disabilitare sia la write cache del disco che quella del sistema operativo, per assicurarsi che le sovrascritture siano effettivamente eseguite sul dispositivo e non nella RAM. Per quanto riguarda la procedura di carving, valgono le stesse considerazioni riportate in 3.3.4.

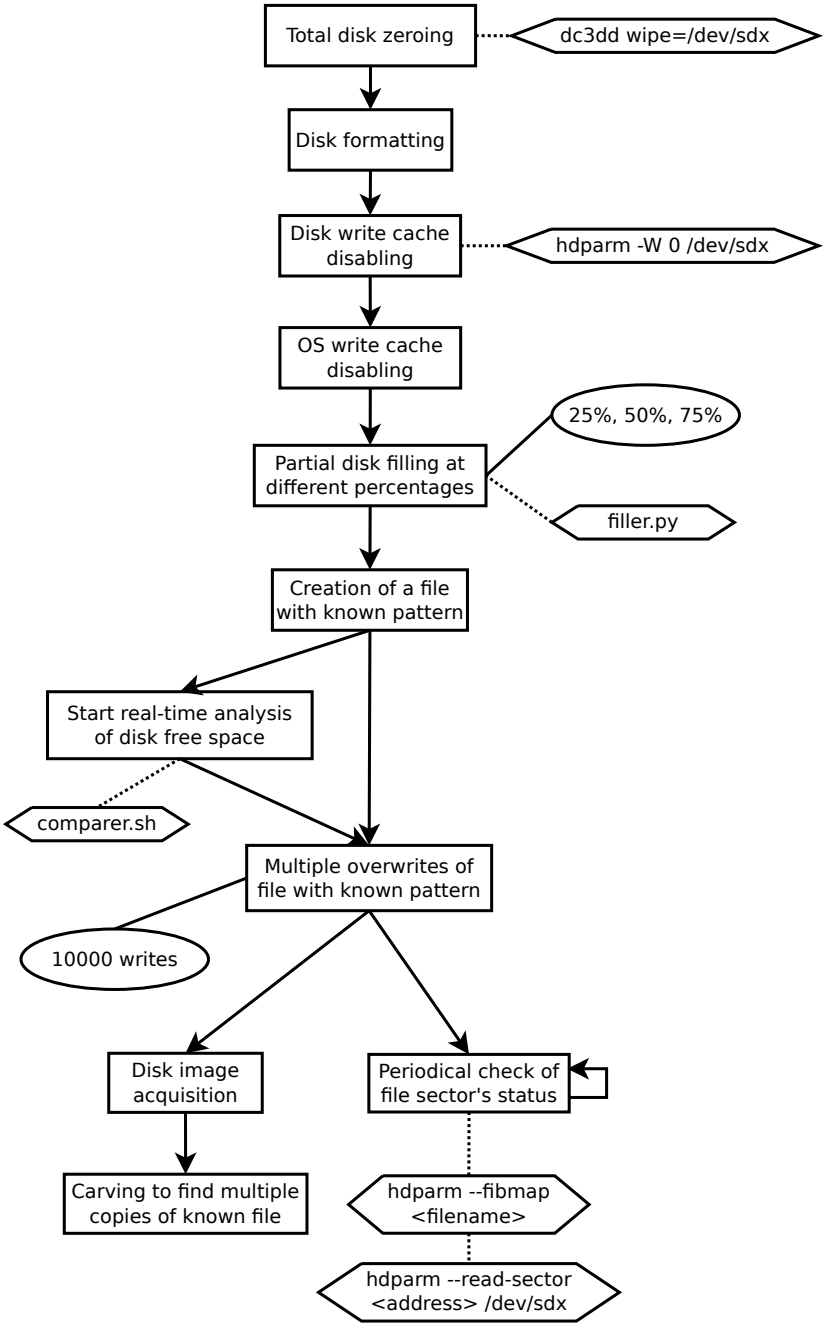


FIGURA 3.6: Protocollo di analisi black box della funzionalità di wear leveling

Informazioni sulle condizioni di attivazione della funzionalità di wear leveling non sono solitamente disponibili, ma dalla documentazione di alcuni produttori e da lavori precedenti ([21], [22], [23]) possiamo dedurre che servano teoricamente almeno due condizioni. La prima è che vi siano blocchi liberi con livelli di scrittura inferiori a quello attuale: per garantire questa condizione abbiamo sempre lasciato almeno il 25% di blocchi liberi durante il riempimento, per coprire anche quei casi in cui il disco non ha un pool di blocchi nascosti da usare. La seconda è che la disparità d'uso tra il blocco attuale e i blocchi liberi superi una certa soglia: 10000 sovrascritture ci è sembrato un numero adatto poichè si tratta di un valore grande raggiungibile in tempi ragionevoli con piccoli file, ma non esiste un'indicazione precisa del valore di questa soglia, che oltre a cambiare con l'evoluzione della tecnologia MLC probabilmente varia per ogni produttore.

L'esito di questo test può fornire un'indicazione di massima della presenza o meno del wear leveling nel disco, ma è preferibile non utilizzarlo per la classificazione. A causa dei mascheramenti effettuati dal controller, è infatti più affidabile utilizzare un test white box per la ricerca degli effetti di questa funzionalità. Il test appena proposto va quindi considerato come un'alternativa da adottare solo laddove non si disponga della strumentazione necessaria per accedere direttamente alle memorie. Si tenga comunque in considerazione che i già citati mascheramenti effettuati dal controller potrebbero, con alta probabilità, rendere del tutto invisibili le operazioni di wear leveling sottostanti.

3.3.7 Ricerca di pattern di azzeramento

Questo test non fornisce un contributo unico alla classificazione del disco, ma può essere molto utile per comprendere nel dettaglio quali sono le politiche di azzeramento implementate nel disco. Infatti finora l'analisi dello spazio libero è stata solo quantitativa, ma questo potrebbe non bastare. Per avere un'idea chiara di cosa succede nel disco potrebbe essere utile costruire una mappa dei settori che vengono modificati dal controller in seguito ad operazioni sui file.

Il test richiede un tempo molto lungo, quindi consigliamo di affiancarlo sempre con un'analisi quantitativa in tempo reale (test su TRIM e garbage collector) e di eseguirlo solo quando si è sicuri che il controller abbia concluso l'esecuzione delle funzionalità di cui si vogliono analizzare gli effetti. Lo scopo principale del test è quello di cercare, in caso di azzeramento parziale dei dati, se il controller segue dei pattern prefissati per decidere quali settori azzerare. Laddove fossero presenti dei pattern, diventa quindi possibile studiare un modo per sfruttarli in modo da nascondere all'interno del disco dati di cui si vuole impedire il recupero o, specularmente, in modo da porre dati importanti in zone del disco dove possono essere sicuramente recuperati.

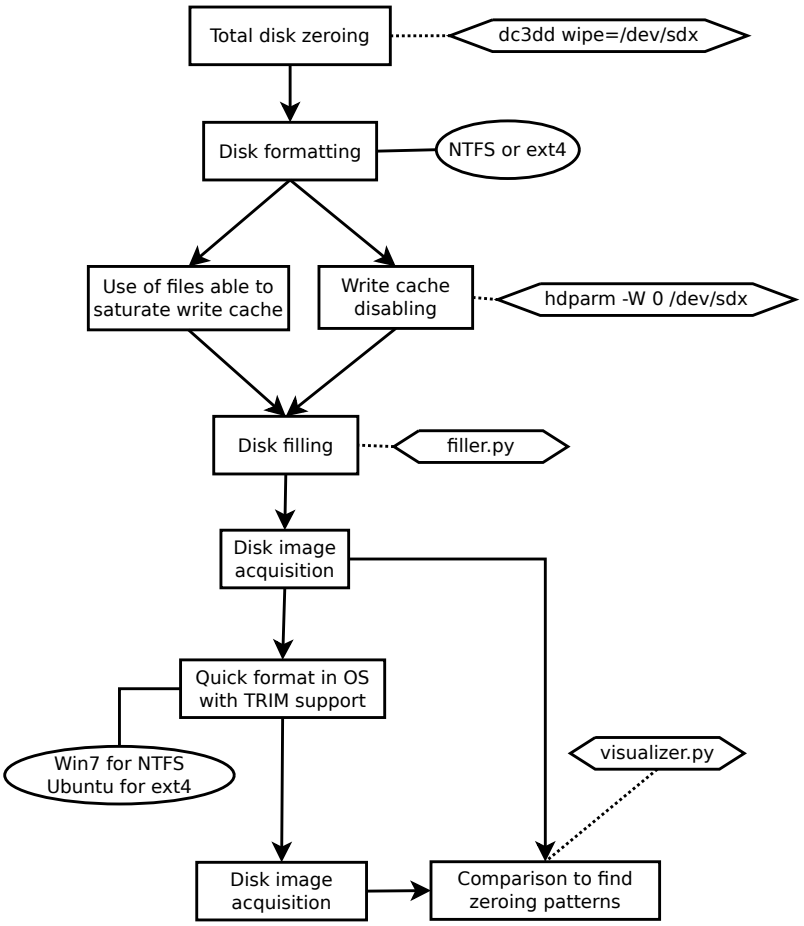


FIGURA 3.7: Protocollo di ricerca di pattern di azzeramento

Il diagramma non contiene una valutazione dell'opportunità o meno di eseguire tale test per evitare ridondanze e perchè, una volta implementata in modo automatizzato, avrebbe comunque comportato un'interruzione dell'esecuzione dello script e la necessità di un intervento dell'utente. Lasciamo quindi all'utente il compito di determinare dai risultati dei test precedenti in quali condizioni l'esecuzione di tale test può essere utile (ad esempio con quali percentuali di riempimento del disco eseguirlo).

3.3.8 Write amplification

L'ultimo test proposto ha l'obiettivo di quantificare il valore di write amplification degli algoritmi implementati nei controller dei dischi a stato solido, misurato come il rapporto tra la quantità di dati scritta fisicamente su disco e la quantità di dati scritta teoricamente dal sistema operativo. L'esito di tale test non influisce sulla classificazione del disco, ma può essere utile per avere una misura di come competono tra loro le funzionalità che puntano ad una riduzione delle scritture su disco (con eventuali effetti negativi sulla recuperabilità dei dati) e quelle che invece portano ad una moltiplicazione dei dati da scrivere (e quindi alla possibilità di recuperare copie multiple dei dati su disco).

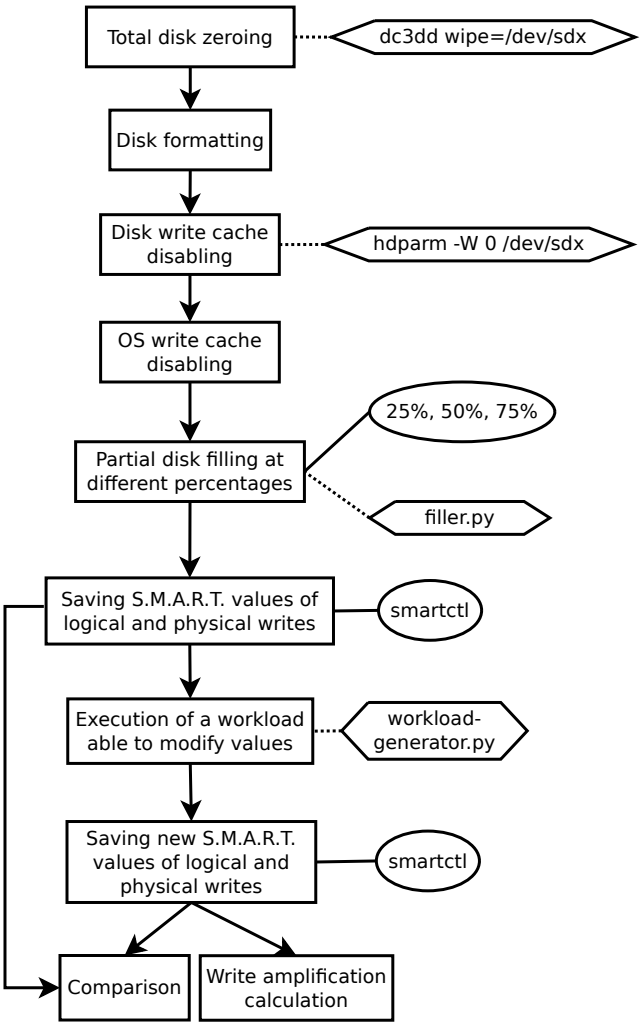


FIGURA 3.8: Protocollo di analisi della write amplification

I fattori che influenzano il valore della write amplification sono numerosi [24]. Tra i più importanti ricordiamo:

- Wear leveling: nel caso in cui non vi siano blocchi liberi disponibili per la sostituzione (ad es. pool di overprovisioning), gli unici blocchi dove è possibile spostare i dati sono blocchi occupati che hanno subito un basso numero di scritture. É quindi necessario spostare i dati contenuti originariamente in questi blocchi meno utilizzati per sostituirli con dati aggiornati frequentemente. Lo spostamento di tali dati porta ad un aumento del numero di scritture fisiche rispetto a quelle logiche.
- TRIM e garbage collection: per quanto detto al punto precedente, la write amplification tende ad aumentare al ridursi dello spazio libero su disco. Funzionalità efficienti di TRIM e garbage collection possono quindi aiutare a mantenere su disco una quantità di spazio libero necessario a ridurre il valore di write amplification.

- Compressione: riducendo il numero di scritture fisiche su disco, il suo effetto porta a ridurre la write amplification verso valori minori di 1.
- Politica di scrittura: se i dati vengono scritti su disco in modo non sequenziale, è possibile che un blocco contenga frammenti di più file. Poichè il blocco è la più piccola unità cancellabile, ogni volta che viene cancellato uno di tali file i frammenti di tutti gli altri vanno riscritti altrove, portando ad un aumento del valore di write amplification.

Laddove i dischi espongano i dati necessari, il valore di write amplification può essere calcolato utilizzando gli attributi S.M.A.R.T. (Self-Monitoring, Analysis and Reporting Technology) dei dispositivi. Questa specifica prevede infatti alcuni campi utili al nostro scopo: il numero di scritture fisiche, il numero di scritture logiche e la differenza di scritture tra il blocco più usato e quello meno usato. Tuttavia la tecnologia ATA S.M.A.R.T. non è standardizzata, quindi ogni produttore espone solo un sottoinsieme di attributi a sua scelta. Non è quindi garantito che i valori necessari ai nostri test vengano resi visibili attraverso il sistema operativo.

Considerata l'eterogeneità dei fattori che influenzano la write amplification consigliamo di eseguire il test su un disco che sia già stato sottoposto a tutti gli esperimenti precedentemente riportati, in modo tale da attivare tutte le funzionalità che possono influenzare il valore cercato e ottenere dei risultati rappresentativi di un uso continuato del disco. Il nostro protocollo prevede in ogni caso l'esecuzione di uno script (workload-generator.py) che sottopone il disco ad un carico di lavoro sufficiente a causare variazioni visibili degli attributi S.M.A.R.T. sotto analisi.

3.4 Protocolli white box

L'introduzione delle tecnologie a stato solido ha portato con sé numerosi problemi riguardanti la recuperabilità ed analizzabilità forense dei dati, causati da una fondamentale limitazione: le tecnologie implementate nei controller che costituiscono il Flash Translation Layer (FTL) creano un livello di astrazione che rende impossibile per l'analista avere una chiara idea della distribuzione fisica dei dati nelle memorie sottostanti. Il Flash Translation Layer infatti mantiene una mappatura fra gli indirizzi dei blocchi logici (LBA), visibili attraverso l'interfaccia ATA, e gli indirizzi dei blocchi fisici (PBA). Ciò che l'analista può realmente vedere attraverso il canale ATA, quindi, è la sola immagine logica dei dati fisici mediata dal controller.

3.4.1 Possibili approcci per il recupero white box dei dati

La possibilità di recuperare, attraverso l'accesso fisico alle memorie, dati non visibili attraverso un normale approccio black box è stata dimostrata da Wei [12]. Tecnologie come il wear leveling infatti, che ha il compito primario di garantire un consumo uniforme di tutte le celle di memoria attraverso la scrittura in blocchi di memoria fisica sempre diversi, hanno come effetto secondario la creazione di copie multiple dei blocchi presenti nel disco. É anche noto che molti dischi in commercio (ed uno dei dischi da noi testati) presentano una quantità di memoria fisica installata maggiore di circa il 10% rispetto alla capacità logica mostrata.

É perciò evidente che l'insieme dei dati recuperabili attraverso un approccio black box sarà sempre un sottoinsieme (al massimo proprio) dei dati fisicamente presenti nelle memorie. Diventa quindi importante, per studiare in modo approfondito l'effettiva recuperabilità dei dati, tentare un approccio di tipo white box, cioè ponendosi al di là delle limitazioni imposte dal controller e accedendo direttamente ai dati immagazzinati nelle memorie. In questo modo è possibile fare leva a proprio vantaggio su tecnologie presenti nel disco, come il già citato wear leveling. Questo approccio non è però facile: i produttori infatti non inseriscono all'interno dei dischi porte JTAG o modalità di debug che possano essere sfruttate da un analista per disabilitare o aggirare il controller.

Nel lavoro di Wei infatti è stato necessario lo sviluppo di un hardware complesso e costoso basato su FPGA [13], che plausibilmente ha richiesto anche un lungo periodo di sviluppo. Lo scopo di questa sezione è quello di fornire una metodologia per il recupero dei dati con un approccio di tipo white box utilizzando strumentazione ove possibile a basso costo, facilmente reperibile e senza le conoscenze necessarie allo sviluppo di un

hardware specifico. La metodologia proposta tuttavia resta valida anche e soprattutto se coadiuvata da hardware specifico e più sofisticato per il recupero dati.

3.4.1.1 Hardware utilizzato

Date le limitazioni di costo considerate, la scelta degll' hardware si è indirizzata verso un programmatore general purpose per chip NAND/NOR chiamato *ProgSkeet*, utilizzato per la programmazione di firmware alternativi sulle memorie NAND di alcuni hardware comuni (come console da gioco), ma mai testato per l'estrazione di dati da dischi a stato solido.

Il primo problema da affrontare nella lettura (dump) di memorie NAND/NOR è la connessione con la piedinatura delle memorie stesse. Il metodo che garantisce la connessione più stabile fra lettore e piedinatura è la rimozione fisica dei chip di memoria dal supporto e la lettura su basette adatte alla piedinatura TSOP48 di cui i chip sono forniti. Questo approccio ha però il grande svantaggio di richiedere una operazione non facilmente reversibile e potenzialmente distruttiva. Queste limitazioni rendono la procedura poco adatta nel caso in cui si vogliano effettuare, come nel nostro caso, numerosi test sugli stessi supporti, e ancor meno adatta all'utilizzo in ambito forense in cui è necessario garantire l' assoluta integrità dei dati. Un'altra via possibile consiste nell'utilizzare una test-clip per creare una connessione alla piedinatura che non richieda la rimozione delle memorie. La reversibilità e quindi ripetibilità di questo approccio è sembrata la migliore scelta possibile per i nostri test e abbiamo così deciso, per ragioni di compatibilità e di costo, di utilizzare una test clip specifica per il lettore in nostro possesso.

3.4.2 Ricostruzione dei dati

Lo scopo del test è verificare se dati contigui a livello logico corrispondono a dati contigui anche a livello fisico o se, come ci aspettiamo, funzionalità implementate nel controller (come il wear leveling) causano frammentazione dei dati a livello fisico.

Il test parte da un azzeramento totale del disco, che consente di ripristinare le condizioni iniziali almeno dal punto di vista dei blocchi logici (l'esatto ripristino dei blocchi fisici del disco non è invece possibile in alcun modo, a meno di non sovrascriverli di zeri attraverso un programmatore di memorie NAND). Vengono quindi scritti sul disco alcuni file di dimensioni differenti composti da blocchi contenenti una fingerprint facilmente distinguibile costituita, ad esempio, da un header riconoscibile, dal logical block address del blocco stesso e da un checksum del contenuto del blocco. Questo procedimento è necessario poichè, passando dal livello logico al livello fisico, la contiguità dei file non

è garantita, e dunque l'utilizzo di file aventi blocchi dal contenuto univoco li rende facilmente riconoscibili, semplificando di molto la ricostruzione dei dati in fase di analisi. Quindi si procede alla lettura delle memorie NAND e alla ricostruzione dei dati riconoscibili grazie alle fingerprint inserite. Infine viene analizzata l'immagine dei blocchi fisici per valutare se a blocchi contigui a livello logico (ad esempio contenenti due frammenti consecutivi di uno stesso file) corrispondono blocchi contigui anche a livello fisico.

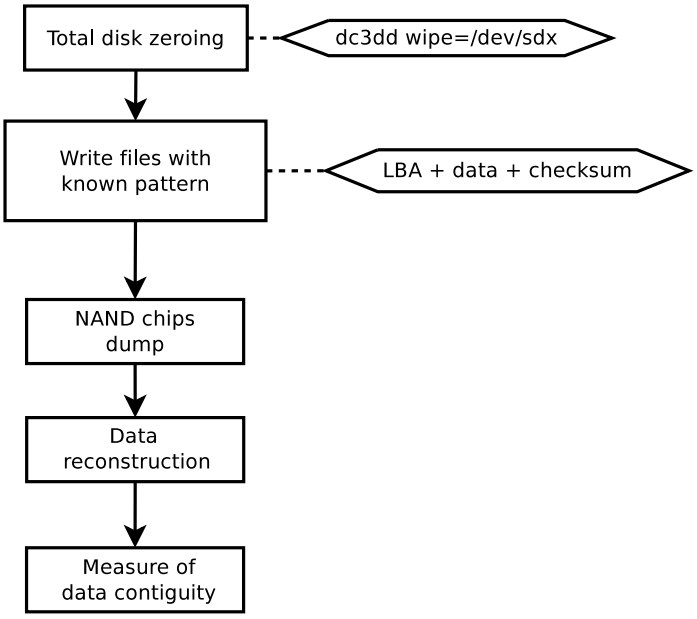


FIGURA 3.9: Protocollo di analisi della possibilità di ricostruire i dati ottenuti attraverso un accesso diretto alle memorie

L'output del test è costituito da tre casi possibili:

- Se i dati sono contigui è possibile applicare gli altri test white box in modo semplice, evitando anche i passi relativi alla ricostruzione dei dati
- Se i dati non sono contigui ma sono in chiaro è possibile applicare gli altri test white box dopo una adeguata procedura di ricostruzione
- Se i dati non sono interpretabili, ad esempio a causa della presenza di cifratura, allora non è possibile ricostruire l'immagine fisica dei dati e nessun test white box può essere svolto; la classe di appartenenza è quindi 3

Occorre aggiungere che, nel caso di dati non interpretabili per cui sia stato verificato che esiste compressione attraverso il relativo test black box (3.3.5), vanno distinti il caso in cui l'incomprensibilità dei dati dipende dalla sola compressione e il caso in cui il dato oltre che compresso viene anche cifrato. Nel primo caso si potrebbe provare ad identificare se viene utilizzato un algoritmo di compressione noto, attraverso la scrittura

di file di dimensione minore alla dimensione di un blocco (quindi non frammentabili) per poi cercare all'interno del disco l'equivalente compresso del blocco iniziale. Una volta identificato l'algoritmo, potrebbe essere infatti possibile ricostruire i dati.

3.4.3 TRIM (white box)

Il test è mirato a verificare se, l'implementazione del comando TRIM, la cui presenza è confermata dai precedenti test black box, è implementato come un semplice mascheramento del controller, che fornisce cioè il byte "0" per ogni richiesta relativa ad un Logical Block Address corrispondente a un file cancellato, oppure corrisponde ad un effettivo azzeramento dei blocchi fisici.

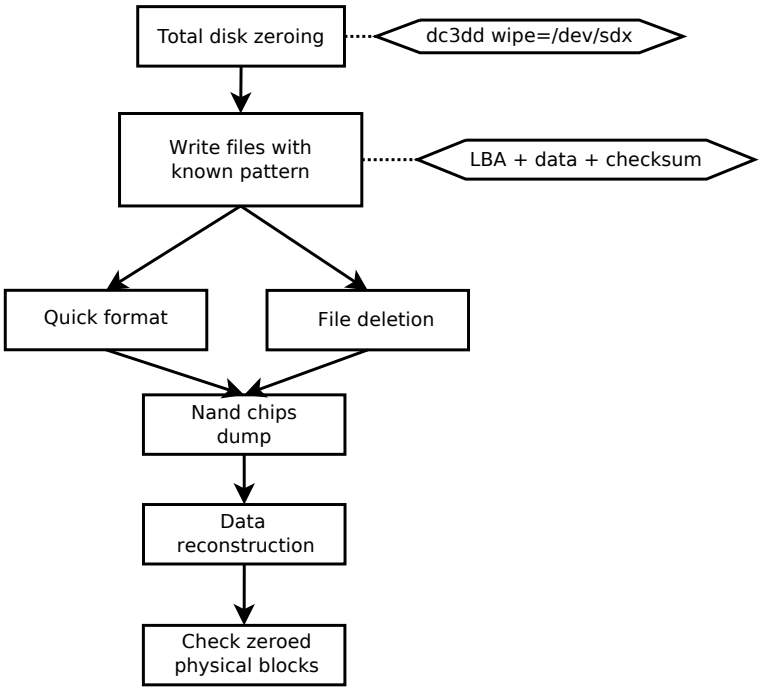


FIGURA 3.10: Protocollo di analisi white box della funzionalità di TRIM

Il test parte anch'esso da un azzeramento totale del disco. Vengono quindi scritti sul disco una sola volta (senza riscritture) alcuni file composti da blocchi contenenti una fingerprint facilmente riconoscibile, per facilitare la ricostruzione dei dati in fase di analisi. A questo punto è possibile seguire due varianti del test: una prevede la normale cancellazione dei file, l'altra un quick format (entrambe devono avvenire in un sistema operativo ed un filesystem che supportino il comando TRIM). Viene quindi effettuata la lettura delle memorie NAND e la ricostruzione dei dati utilizzando la fingerprint presente nei blocchi. Si cercano quindi i blocchi corrispondenti ai file cancellati. A seconda dei risultati, le possibili classificazioni sono:

- Se i blocchi sono presenti significa che il TRIM non comporta una vera cancellazione fisica, ma invece essa viene "simulata" dal controller attraverso un mascheramento. I dati sono quindi recuperabili e la classe di analizzabilità del disco è 1
- Se invece non sono presenti, allora il TRIM corrisponde ad una effettiva cancellazione fisica. La classe del disco potrebbe quindi essere 2

Per la classe 2 è stata usata una forma ipotetica poichè, per definire la recuperabilità come pari al caso black box, è necessario che il medesimo risultato venga fornito anche dai test white box sul wear leveling (3.4.4).

3.4.4 Wear leveling (white box)

Lo scopo del test è verificare se, dopo aver simulato uno scenario di normale utilizzo, possono essere recuperati dati non visibili attraverso approcci white box, e in particolare se sono presenti copie multiple dei file presenti all'interno delle memorie. Il numero di repliche del file è un indice complesso, direttamente proporzionale alla write amplification ed inversamente proporzionale all'efficacia dei meccanismi di garbage collection utilizzati dal controller del disco. Ponendoci dal punto di vista della analizzabilità forense abbiamo deciso di semplificare la classificazione del risultato del test limitandoci a considerare la presenza o assenza di copie multiple dei file. Effettuare questo test ci consente di avere una valutazione diretta della possibilità che un file sia recuperabile all'interno del disco stesso, e quindi dell'analizzabilità forense del dispositivo attraverso approcci white box.

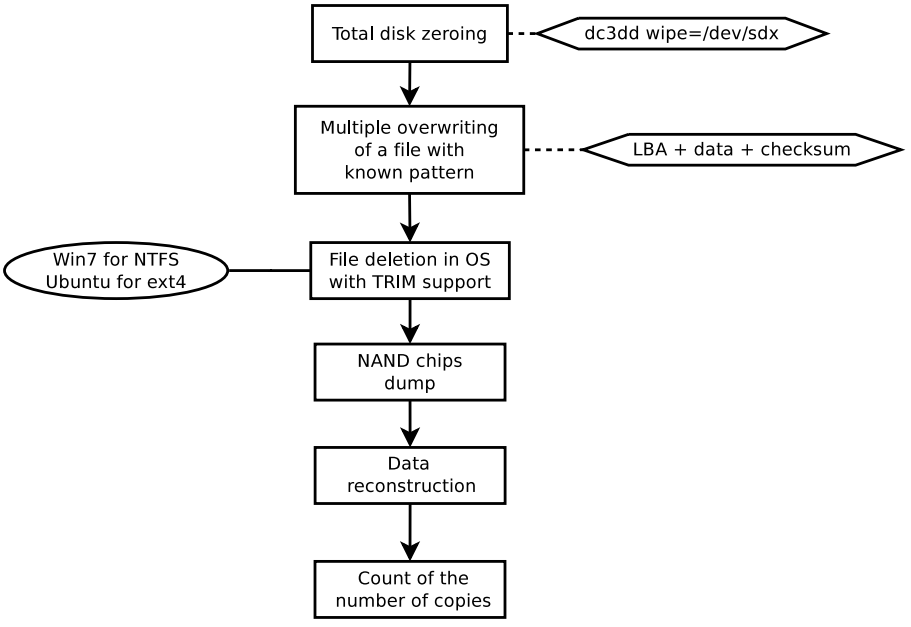


FIGURA 3.11: Protocollo di analisi white box della funzionalità di wear leveling

Come il precedente, anche questo test inizia con l'azzeramento totale del disco. In seguito vengono scritti sul disco alcuni file composti da blocchi contenenti una fingerprint facilmente riconoscibile per facilitare la ricostruzione dei dati in fase di analisi. Si procede quindi alla cancellazione dei file utilizzando un file system e un sistema operativo che supportino le operazioni di TRIM, alla lettura delle memorie NAND e alla ricostruzione dei dati sfruttando l'univocità dei blocchi scritti in precedenza. A questo punto è possibile verificare la presenza di repliche dei file scritti sul disco e la loro integrità grazie al checksum contenuto in ogni blocco.

Interpretazione dei Risultati

L'output del test è binario, corrispondente alla presenza o assenza di dati non recuperabili attraverso un approccio black box.

- Se possono essere recuperati dati non visibili attraverso un'analisi black box la classe è 1
- Se non può essere recuperato alcun file che non sia già visibile attraverso i normali approcci black box la classe potrebbe essere 2. Le spiegazioni possibili sono due: o non avvengono fenomeni di wear leveling (molto improbabile), oppure gli algoritmi di garbage collection sono molto efficienti e si occupano di cancellare dal disco tutte le copie fisiche dei file nel momento in cui essi vengono cancellati

La classe 2 è anche qui in forma ipotetica perchè, come detto in 3.4.3, per la classificazione è necessario che il medesimo risultato sia confermato dal test sul TRIM.

3.4.5 Garbage collection (white box)

Questo test si propone di verificare in modo esatto come agisce e quali alterazioni dei dati causa il garbage collector. Le ipotesi sul funzionamento dei meccanismi di garbage collection all'interno dei controller sono diverse: da una parte si ritiene il garbage collector un sistema capace soltanto di azzerare in modo autonomo settori del disco non utilizzati, attraverso la lettura della file allocation table del filesystem; dall'altra si sostiene invece che il garbage collector compia anche operazioni di ottimizzazione attraverso, ad esempio, l'unione di due blocchi sotto utilizzati in un unico blocco. Nessuna di queste ipotesi però è ancora stata studiata in modo approfondito, poichè tali meccanismi non hanno effetti visibili sui blocchi logici del disco, ma soltanto sui blocchi fisici, escludendo così la possibilità di effettuare esperimenti con normali approcci black box. Un solo esperimento documentato, fino ad oggi disponibile, ha mostrato cambiamenti evidenti dei blocchi logici (e non soltanto fisici) riconducibili a processi di garbage collection [10], ma la

mancanza di altre conferme e l'impossibilità di ottenere i medesimi risultati replicando in modo esatto gli esperimenti ci porta a credere che si tratti di un caso isolato o ad errori nello svolgimento degli esperimenti.

Con questo test è possibile verificare quale delle due ipotesi sopra citate è vera. É plausibile aspettarsi che i risultati di questo esperimento siano differenti per ogni produttore e dipendenti dal singolo modello di controller, ed è sicuramente possibile che per alcuni controller entrambe le ipotesi possano essere vere contemporaneamente.

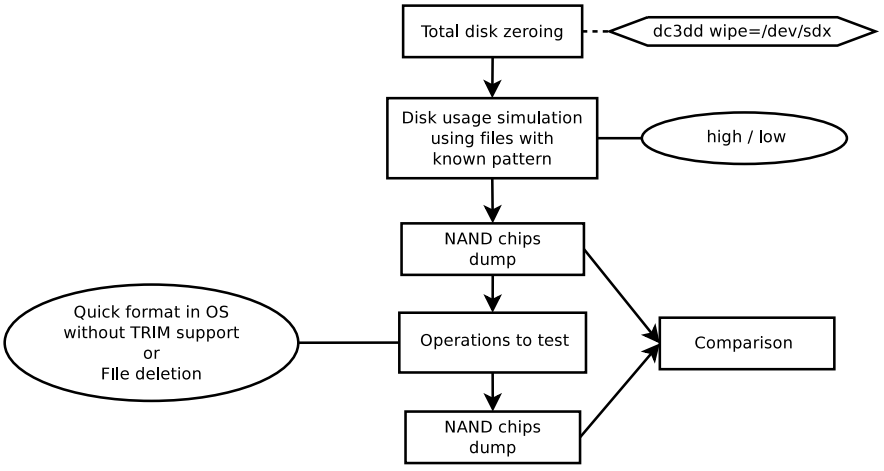


FIGURA 3.12: Protocollo di analisi white box della funzionalità di garbage collection

Il test parte anche in questo caso da un azzeramento totale del disco. In seguito vengono scritti sul disco alcuni file composti da blocchi contenenti una fingerprint facilmente riconoscibile, esattamente come nel test precedente. Si effettua quindi una prima lettura delle memorie NAND, salvando così lo stato iniziale della memoria fisica del disco. In seguito possono essere svolte varie operazioni, quali il quick format su un sistema operativo che non supporti TRIM (così che eventuali azzeramenti riscontrati non possano essere attribuiti a comandi di TRIM inviati dal sistema operativo) oppure cancellazioni di singoli file in diversi stati del disco (ad esempio con diverse percentuali di riempimento). Effettuata l'operazione che si vuole analizzare, si procede ad una seconda lettura delle memorie NAND ottenendo lo stato finale della memoria fisica del disco. Infine le due immagini ottenute vengono comparate per costruire una mappa di spostamento e cancellazione dei blocchi.

Interpretazione dei Risultati

L'output del test non partecipa alla classificazione perchè si limita a costruire una immagine del disco che ne rappresenta le differenze fra le condizioni precedenti e seguenti alle operazioni testate. Attraverso questa immagine è immediato verificare le ipotesi di azzeramento od ottimizzazione dei dati. I possibili esiti dell'esperimento sono quindi i seguenti:

- Non viene rilevata nessuna operazione di garbage collection
- Il garbage collector causa azzeramento di blocchi fisici
- Il garbage collector causa una ottimizzazione dei blocchi fisici
- Il garbage collector causa sia azzeramento dei blocchi che ottimizzazione dei blocchi fisici

3.4.6 Problemi pratici nell'analisi white box

Nella sperimentazione reale dei protocolli, ci siamo imbattuti in numerosi problemi pratici di natura elettronica che che qui descriviamo. Il primo problema è dovuto all'utilizzo di una test clip per evitare la necessità di dissaldare i chip di memoria. L'utilizzo di una test clip ha però mostrato grossi limiti nella stabiltà della connessione. Una connessione instabile ha significato non solo l'acquisizione di dump non significativi, ma grandi difficoltà nella configurazione dei parametri corretti per la lettura delle memorie. Ogni chip è infatti caratterizzato da un numero totale di pagine, un numero di blocchi per pagina, e dalla durata dei cicli di lettura e scrittura, informazioni spesso non rilasciate dal produttore. Inoltre il throughput effettivo del lettore, di circa 1MB/s, ha reso il tempo necessario per la lettura di un singolo chip di memoria superiore a un'ora, tempo maggiore di qualunque durata di una connessione stabile attraverso test clip. I driver e i software utilizzati dalla piattaforma per interfacciarsi al sistema operativo sono proprietari, quindi la scrittura di un software alternativo o la modifica di quello esistente sarebbero operazioni estremamente complesse.

Un ulteriore problema affrontato durante i tentativi di lettura dei chip NAND è stato l'attivazione involontaria del controller. Le linee di alimentazione infatti sono condivise fra memorie e controller, ed ogni tentativo di alimentare una memoria comporta l'accensione del controller e le conseguenti operazioni di lettura e scrittura che si sovrappongono al tentativo di lettura della singola memoria causando errori. Per ovviare al problema si è pensato di disattivare il controller inserendo al posto della normale resistenza in serie dell'oscillatore una resistenza removibile, così da poter fermare il clock ed impedire che qualunque attività fosse compiuta dall'oscillatore.

Nonostante i numerosi tentativi e le differenti soluzioni testate, non siamo stati in grado, con l'hardware in nostro possesso, di ottenere alcuna lettura completa e leggibile delle memorie NAND dei dischi a nostra disposizione. La spiegazione può essere trovata in una somma di fattori che sono: l'impossibilità di avere una connessione stabile con le memorie a causa della scarsa qualità delle test clip reperibili sul mercato, la presenza di software closed source e la difficoltà di avere informazioni sui protocolli di lettura e scrittura delle

singole memorie, causato dalla mancanza di datasheet pubblici. In particolare per i dischi Samsung non ci sarebbe nessuna possibilità di effettuare letture senza la rimozione fisica del chip, poiché utilizzano una piedinatura a pin nascosti che non consente di utilizzare una test clip.

I numerosi problemi riscontrati ci indicano che per accedere con successo a un chip NAND è necessario lo studio approfondito dei protocolli di lettura e scrittura delle singole memorie usate dai produttori e la realizzazione di un lettore ad-hoc, ad esempio basato su FPGA, che garantisca prestazioni di lettura elevate e un alta qualità della connessione con la piedinatura delle memorie, caratteristiche che porterebbero il costo del prodotto ben oltre al limite di quella che può essere considerata una soluzione a basso costo. I problemi da affrontare quindi sono molteplici e richiedono competenze che non fanno parte della preparazione di un comune analista forense. Infine la presenza in controller come il Sandforce SF-1200 di meccanismi di compressione e cifratura in hardware renderebbe vano il tentativo di ricostruzione dei dati, che risulterebbero inutilizzabili.

Nonostante non sia stato possibile effettuare i test, la metodologia proposta rimane valida e può essere applicata con successo disponendo di hardware specifico con cui superare i problemi di natura elettronica riscontrati.

3.5 Un riassunto delle associazioni tra metriche e test

Per concludere il capitolo forniamo un breve riassunto delle associazioni tra le metriche di classificazione e i test necessari per scegliere la classe da associare ai dispositivi (che varia in base al file system utilizzato e al sistema operativo a cui il disco è interfacciato).

Recuperabilità con Metodi Black Box

É necessario effettuare almeno due test, ovvero il test sul TRIM (3.3.3) e il test sull'aggressività del garbage collector (3.3.2), e basarsi sul comportamento di azzeramento più aggressivo rilevato tra i due, così da prendere in considerazione per la classificazione il loro effetto combinato. Se viene rilevato un azzeramento totale questi due test sono sufficienti, ma in caso contrario è consigliato verificare se i dati sono effettivamente recuperabili in forma integra eseguendo il test di recuperabilità (3.3.4).

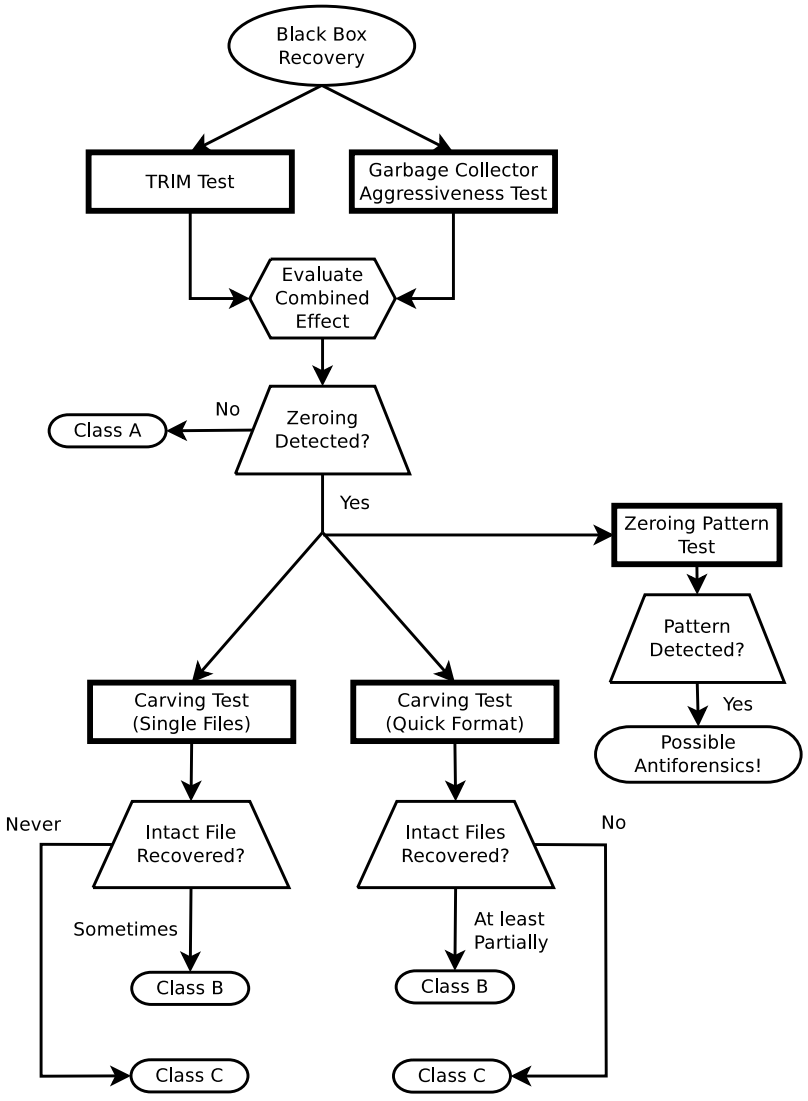


FIGURA 3.13: Associazioni tra metriche di recuperabilità black box e test

Come mostrato in Figura 3.13, in base ai risultati dei test le classi sono le seguenti:

- Se non viene rilevato alcun azzeramento, classe A.
- Se nonostante l'effetto combinato di entrambi i test è garantita una recuperabilità parziale, classe B.
- Se in seguito all'azione di entrambi i test vi è azzeramento totale, classe C.

Integrità dei dati

Il test da effettuare è uno solo, ovvero il test di stabilità dell'hash (3.3.1). Se il confronto tra hash indica un cambiamento, la classe di utilizzabilità dei dati come prove è $-$, altrimenti se l'hash non cambia la classe è $+$.

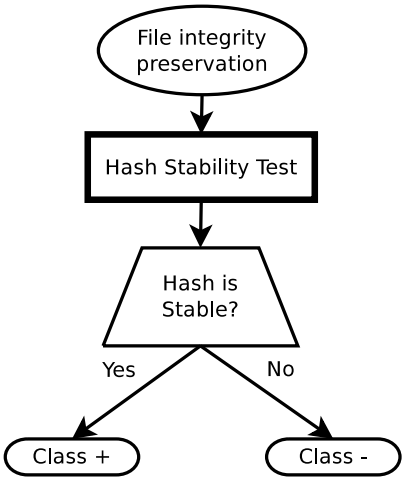


FIGURA 3.14: Associazioni tra metriche di preservazione dell'integrità dei dati e test

Recuperabilità con Metodi White Box

Il primo passo da eseguire per la classificazione non corrisponde ad un test, e consiste nel verificare se le memorie NAND hanno una piedinatura in grado di interfacciarsi con gli strumenti di lettura delle memorie. Nel caso in cui questa condizione non fosse verificata, la classe è 4. Se la piedinatura è invece accessibile, allora è necessario eseguire dei test. Il primo test da effettuare è quello per analizzare se è presente una funzionalità di compressione implementata a livello di controller (3.3.5). Nel caso in cui la funzionalità sia presente la classe è 3, altrimenti si può procedere con la lettura delle memorie per analizzare la possibilità di ricostruire i dati. Se i dati estratti risultano incomprensibili questo esito potrebbe indicare che viene eseguita una cifratura automatica dei dati in fase di scrittura e il disco ricadrebbe nella classe 3, ma se invece i dati risultano ricostruibili vanno eseguiti due test: il test di analisi white box della funzionalità di TRIM (3.4.3) e il test di analisi white box della funzionalità di wear leveling (3.4.4).

In base ai risultati dei test, le classi da associare al dispositivo sono le seguenti:

- Se almeno uno dei test riesce a recuperare dati integri non visibili con la precedente analisi black box, la classe da associare al dispositivo è 1.
- Se invece i dati integri recuperati da entrambi i test sono tutti ottenibili anche con un'analisi di tipo black box, la classe è 2.

Il diagramma 3.15 riassume graficamente i passi da seguire per classificare il disco secondo le metriche di analizzabilità white box.

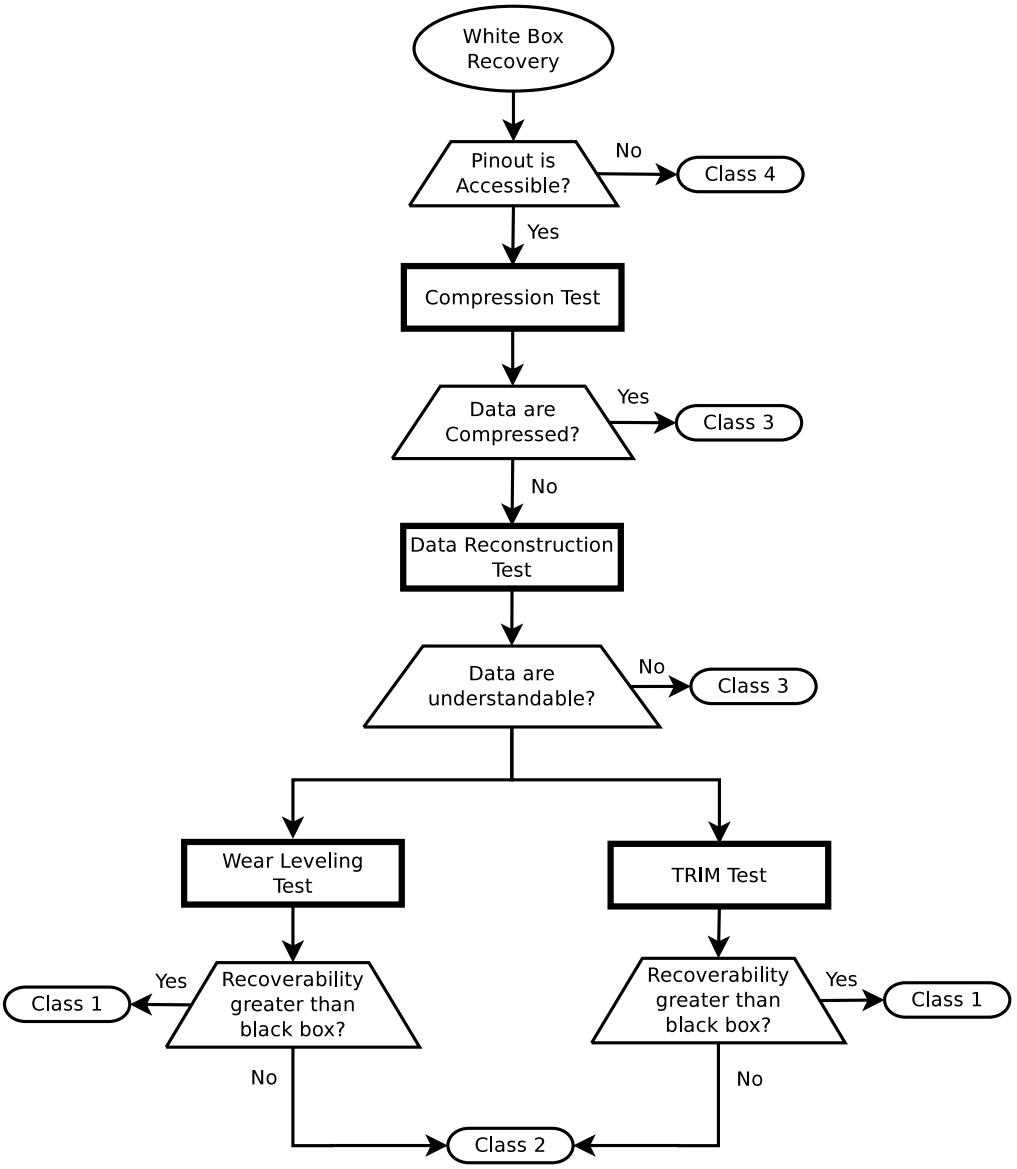


FIGURA 3.15: Associazioni tra metriche di analizzabilità white box e test

4. Protocolli black box: un esempio di applicazione

In questo capitolo mostriamo come abbiamo applicato ai dischi in nostro possesso i protocolli black box proposti, illustrando alcuni esempi di possibili risultati e fornendo le nostre interpretazioni in merito ad essi. I dischi usati per gli esperimenti sono un Corsair F60 (Controller Sandforce SF-1200), un Samsung S470 MZ-5PA064A (Controller Samsung ARM base 3-core MAX) e un Crucial M4 (Controller Marvell 88SS9174-BLD2). Il computer a cui sono stati collegati è un Dell OPTIPLEX 755 con processore Intel Core 2 Quad CPU Q9300 @ 2.5 GHz e 4GB di RAM. Come sistemi operativi abbiamo usato Microsoft Windows 7 Professional SP1, Ubuntu 12.04 e DEFT 7.
Ognuno dei protocolli di testing è stato implementato sotto forma di script bash, i quali utilizzano nel corso dell'esecuzione i tool open source citati nei diagrammi dei protocolli, oltre ad alcuni tool sviluppati da noi in bash o in python. Il codice delle implementazioni dei test e dei nostri tool può essere consultato in appendice a questo lavoro. I test di questo capitolo sono stati eseguiti utilizzando tali script. Per i test sotto Windows, gli script sono stati eseguiti attraverso Cygwin.
Al termine del capitolo forniamo infine, per ognuno dei dischi testati, la classificazione in base alle metriche di analizzabilità black box che abbiamo proposto.

4.1 Garbage collection - Stabilità hash e aggressività azzeramento

Per testare la presenza di un ipotetico garbage collector implementato a livello di controller abbiamo seguito tutte le diramazioni dei due protocolli illustrati, mantenendo i dischi collegati al computer anche per un tempo superiore a quello indicato, in un caso arrivando anche a 60 ore. In nessun caso è stata rilevata una variazione dello spazio occupato su disco.
Per quanto riguarda la stabilità dell'hash in fase di idle, sebbene sotto sistemi di uso quotidiano come Windows 7 e Ubuntu sia stato registrato un prevedibile cambiamento,

gli esperimenti più importanti, quelli sotto DEFT, hanno lasciato l'hash di tutti i dischi completamente immutato. Ipotizziamo quindi che i cambiamenti rilevati nei primi due sistemi operativi siano dovuti ad attività di background eseguite dal sistema operativo, ad esempio a fini di ottimizzazione. Poichè l'hash non è cambiato durante gli esperimenti sul sistema operativo ad uso forense, abbiamo ipotizzato che la spiegazione possa essere una delle seguenti:

- La funzionalità di garbage collection non è realmente implementata
- La funzionalità di garbage collection è strettamente legata all'azione del comando di TRIM, poichè il controller non è in grado di distinguere autonomamente le peculiarità di ogni file system, e dunque non può funzionare su un sistema operativo che non lo supporti come una distribuzione per uso forense.
- L'azione del garbage collector è mascherata dal controller e può essere analizzata solo con metodologie di tipo white box

Samsung fornisce un software per eseguire operazioni di garbage collection sui propri dischi anche in sistemi privi del supporto TRIM. Questo software si chiama SSD Magician, è closed source ed è disponibile solo per Windows. Lo abbiamo testato sotto Windows XP SP3, seguendo la solita procedura di riempimento a diverse percentuali e quick format, lanciando lo script di analisi dello spazio libero su disco e poi eseguendo attraverso il software l'operazione di garbage collection. Nonostante la solita attesa di 16 ore, lo script non ha rilevato alcuna variazione nella percentuale di settori occupati.
Poichè i nostri risultati risultavano in contrasto con quanto scoperto in [10], abbiamo inoltre voluto riprodurre l'esperimento di Bell e Boddington utilizzando disco, firmware, sistema operativo e script indicati nel loro lavoro. Nemmeno in questo modo siamo riusciti a rilevare azzeramenti di dati in seguito a un quick format. Abbiamo contattato gli autori del lavoro ma neanche in questo modo siamo riusciti a identificare la causa di questa differenza.

4.2 TRIM

L'esecuzione dei test relativi alla funzionalità di TRIM ha dato risultati molto vari. Il fatto che un disco disponga del supporto per questo comando, infatti, non è sufficiente ad identificare uniformemente come vengono trattati i file da cancellare. Il comportamento effettivo di questo comando può dipendere da vari altri fattori, quali il sistema operativo, il file system utilizzato, i driver e/o il firmware. Nella nostra analisi ci siamo concentrati soprattutto su file system e sistemi operativi. In particolare abbiamo analizzato il comportamento sotto NTFS ed ext4, in quanto si tratta di due dei file system più diffusi.

4.2.1 Filesystem NTFS

Attualmente Windows 7 è l'unico sistema operativo che supporta la funzionalità di TRIM per file system NTFS, e solo per dispositivi in modalità AHCI. I nostri esperimenti hanno confermato che il comportamento di NTFS-3G, usato in sistemi Linux, è radicalmente diverso da quello dell'implementazione proprietaria di NTFS, in quanto non presenta alcun effetto legato alle funzionalità peculiari degli SSD. Gli esperimenti che verranno esposti in questa sezione sono quindi stati realizzati con il sistema operativo di Microsoft.

Seguendo con il nostro script tutte le diramazioni del test proposto, abbiamo notato che i dischi possono essere suddivisi in due categorie: dischi ad azzeramento totale e ad azzeramento parziale.
I dischi ad azzeramento totale (Samsung S470 e Crucial M4) hanno mostrato un comportamento estremamente aggressivo sia nell'azzeramento di singoli file che in caso di quick format. Per quanto riguarda i singoli file, dopo l'invio del comando di cancellazione i settori ospitanti il file vengono azzerati in un tempo che va da un minimo di 5 secondi per il disco Samsung a un massimo di 10 secondi per il disco Crucial. Analoga è la situazione in caso di formattazione. Per quanto riguarda il modello Samsung S470, riempiendo tutto lo spazio disponibile l'intero contenuto del disco viene azzerato entro un massimo di 10 secondi dalla conclusione del quick format. Il Crucial M4 tuttavia riesce ad essere ancora più rapido nell'azzeramento: il controller comunica la rimozione completa del contenuto del disco prima ancora che il sistema operativo consideri terminata la formattazione. La recuperabilità dei dati con un'analisi black box è quindi completamente compromessa per questi due dischi, con conseguenze molto negative dal punto di vista dell'analisi forense.
Il disco Corsair si comporta invece in modo decisamente differente, ricadendo nella classe dei dischi ad azzeramento parziale. In seguito all'esecuzione di un quick format, soltanto una ridotta percentuale di dati su disco viene azzerata. Ripetuti esperimenti hanno rivelato che questa percentuale è proporzionale alla quantità di spazio libero su disco. La Figura 4.1 mostra la dipendenza tra le percentuali di spazio libero iniziale (asse delle ascisse) e di spazio azzerato (asse delle ordinate) sul disco Corsair. É possibile notare che esistono 5 intervalli di valori in cui la quantità di spazio azzerato aumenta linearmente, mentre i valori restanti corrispondono a delle soglie in cui la quantità di settori azzerati resta costante.
Anche i test sulla cancellazione di singoli file hanno dato risultati inaspettati. In seguito ad alcune cancellazioni seguiva un azzeramento dei settori contenenti il file entro 3 secondi circa dall'invio del comando, ma altri file invece non venivano azzerati e restavano teoricamente recuperabili. Abbiamo eseguito l'esperimento a diverse percentuali di riempimento del disco, cercando di scoprire se questo comportamento fosse dipendente

dalla quantità di spazio libero sul dispositivo. Il test è stato eseguito in cinque condizioni diverse, prima con un disco contenente solo il file da cancellare e successivamente riempiendo il disco al 25%, 50%, 75% e 100% della sua capacità. Indipendentemente dalla percentuale di spazio occupato, il verificarsi dell'uno o dell'altro comportamento sembrava alternarsi con un criterio pressoché casuale. Per indagare più dettagliatamente le politiche di azzeramento dietro ai singolari comportamenti di questo disco, abbiamo quindi deciso di eseguire un test di ricerca di eventuali pattern di azzeramento (vedi Sezione 4.3).

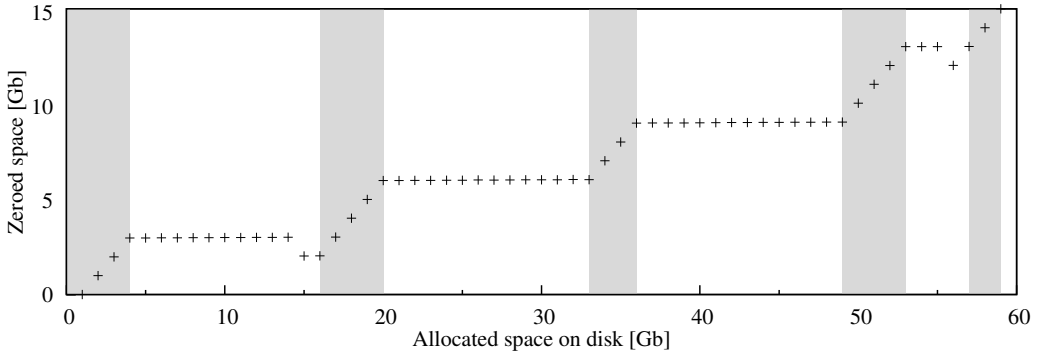


FIGURA 4.1: La quantità di blocchi azzerati dalla funzionalità di TRIM sul disco Corsair F60 con file system NTFS dipende dalla quantità di spazio utilizzato.

4.2.2 Filesystem ext4

Come già accennato, per testare il comportamento della funzionalità di TRIM con ext4 abbiamo usato Ubuntu 12.04, in quanto solo il kernel Linux supporta il comando sotto questo file system. La funzionalità non è tuttavia presente di default, ma va attivata manualmente modificando il file /etc/fstab: in particolare, bisogna aggiungere *discard* alla lista delle opzioni per il disco a stato solido. Una volta abilitato il TRIM, abbiamo seguito tutte le diramazioni del protocollo proposto implementate nel nostro script.

Per quanto riguarda il quick format i risultati sono stati abbastanza uniformi, probabilmente dovuti all'uso per ogni disco del driver generico AHCI. In tutti i casi il nostro analizzatore in tempo reale ha rilevato un azzeramento totale dell'intero contenuto del disco entro un massimo di 15 secondi dall'inizio della formattazione. Si tratta di un risultato estremamente negativo, poichè impedisce completamente la recuperabilità dei file con un approccio black box.
Per quanto riguarda la cancellazione di singoli file, invece, sono emersi risultati molto diversi da disco a disco:

- Corsair F60: I file vengono sempre azzerati, ad una velocità intorno a 1 GB/s. Non è stato quindi rilevato alcun comportamento di azzeramento parziale simile a quelli notati sotto NTFS. Probabilmente i driver proprietari o il controller Sandforce implementano delle funzionalità specifiche per il file system NTFS.

- Samsung S470: I singoli file cancellati non vengono azzerati. L'analizzatore in tempo reale rileva tuttavia una variazione dello spazio libero su disco pari allo 0.01%. La nostra ipotesi è che sia dovuta all'aggiornamento di qualche metadato.

- Crucial M4: In seguito ad una cancellazione apparentemente non avviene alcun azzeramento di file, ma non appena il disco viene smontato lo script rileva entro 5 secondi un aumento dello spazio azzerato su disco pari alla dimensione del file cancellato. Ipotizziamo quindi che l'azzeramento del file sia immediato, ma la tabella dei file venga aggiornata solo in fase di unmount e quindi il controller continui a comunicare la presenza dei file cancellati finchè il disco è montato.

Quest'ultimo comportamento è stato rilevato anche in una situazione che esula dallo scopo del test, ovvero la scrittura di nuovi file. In tutti i dischi, quando un nuovo file viene scritto su disco l'analizzatore in tempo reale non rileva alcuna diminuzione dello spazio libero. Tuttavia, non appena il disco viene smontato, l'aumento di spazio occupato viene rilevato istantaneamente. Questo dimostra che nel caso del file system ext4 è presente un alto livello di mascheramento del contenuto del disco dovuto al controller.

4.3 Ricerca di pattern di azzeramento

La presenza di gradini nel grafico in Figura 4.1 ci ha fatto ipotizzare che per il disco Corsair, con file system NTFS, esistessero fasce di valori per cui la percentuale di azzeramento fosse identica. Questo ci ha portato a supporre la presenza di pattern di azzeramento fissi. Abbiamo quindi sviluppato uno script per ottenere una rappresentazione grafica di come ogni settore del disco (corrispondente a un pixel dell'immagine) viene modificato in seguito ad un quick format. Eseguendo questa visualizzazione a diverse percentuali di riempimento, abbiamo scoperto che l'azzeramento avviene sempre in fasce di settori prefissate, come mostrato in Figura 4.2.
Tutte le fasce sono di 4.294.901.760 byte, esclusa la prima, il cui spazio azzerabile è di poco inferiore (4.261.347.328 byte). Ciò è dovuto al fatto che questa fascia contiene alcuni importanti metadati, in particolare la copia della Master File Table (MFT). Il nostro script di visualizzazione, infatti, rileva in seguito ad ogni formattazione una modifica dei valori di alcuni settori presenti in questa fascia che non corrisponde ad un azzeramento. In Figura 4.3 è possibile vedere questi settori rappresentati con il colore rosso.

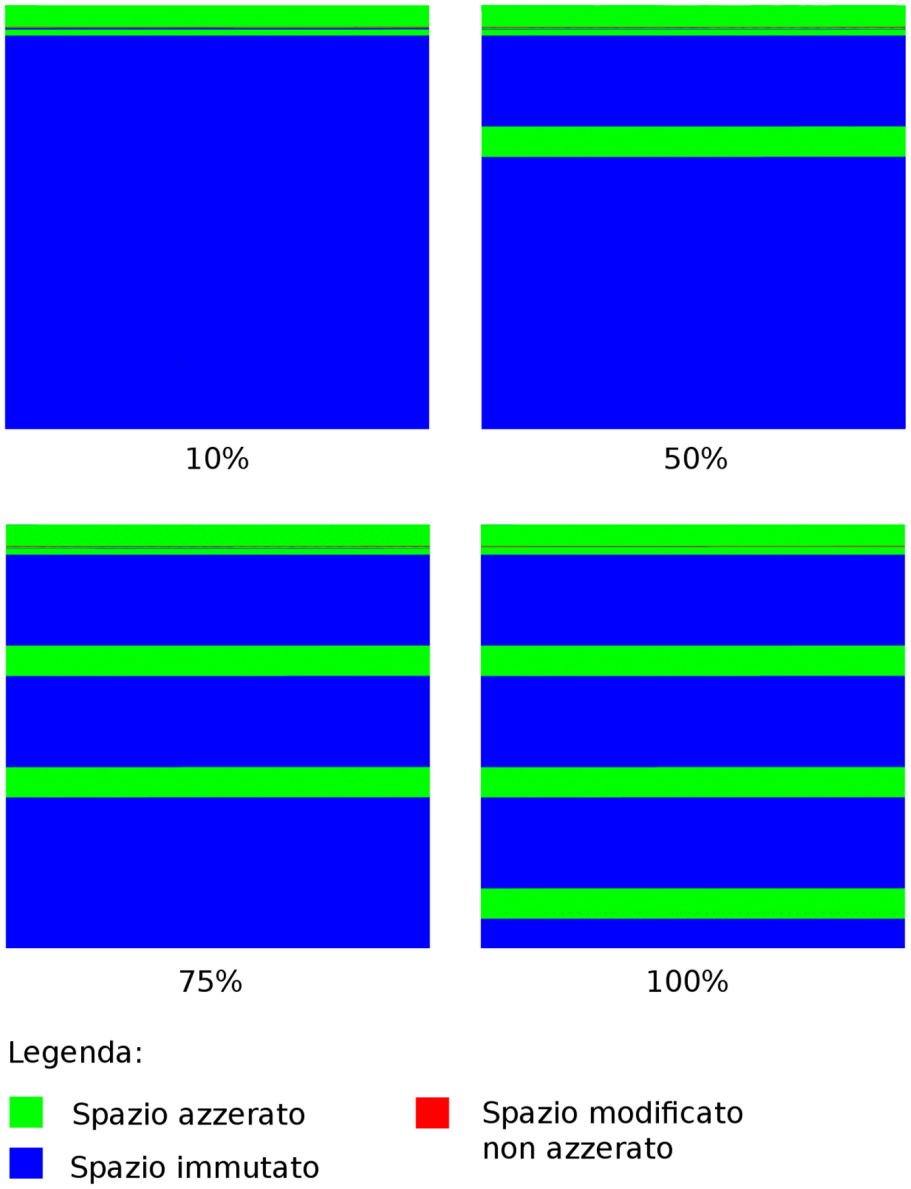


FIGURA 4.2: Confronto tra le zone di azzeramento del disco Corsair (file system NTFS) a diverse percentuali di riempimento. Le zone sono costanti e il loro numero dipende dalla quantità di dati su disco.

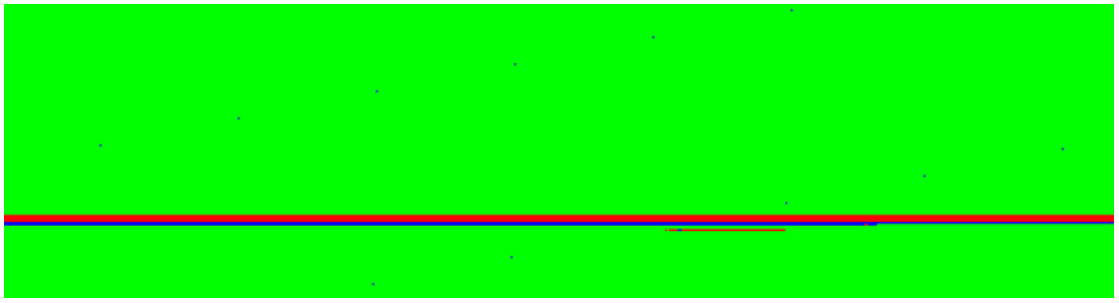


FIGURA 4.3: Un ingrandimento di parte della prima fascia di azzeramento del disco Corsair (file system NTFS). Nei settori indicati in rosso è presente la copia della MFT.

Per quanto riguarda la politica di azzeramento legata alla cancellazione di singoli file, inoltre, verificando con hdparm gli indirizzi dei settori dei file cancellati abbiamo scoperto che anch'essa è collegata alle fasce di azzeramento. Se il file da cancellare è situato dentro una delle fasce note, i settori che lo ospitano vengono azzerati entro 3 secondi circa dall'invio del comando di cancellazione. Al contrario, se il file si trova al di fuori delle fasce, in seguito ad una cancellazione da sistema operativo non viene rilevato alcun aumento dello spazio libero su disco e hdparm indica i settori come ancora pieni.

4.3.1 Come sfruttare i pattern trovati

Una volta scoperta la presenza di pattern fissi, ci siamo chiesti se fosse possibile utilizzarli per garantire oppure ostacolare la recuperabilità di alcuni file di nostra scelta. In particolare, è possibile scrivere e leggere dati dentro a settori di nostra scelta, rendendoli eventualmente invisibili attraverso una normale analisi del file system? Abbiamo perciò implementato una proof of concept per dimostrare che la presenza di fasce fisse di azzeramento può essere sfruttata da tecniche di antiforense.

Il primo passo che l'utente deve compiere è eseguire attraverso lo script un'operazione preliminare da fare una volta sola, a disco vuoto. L'idea è quella di scrivere all'interno di ognuna delle fasce di azzeramento un file che occupi tutti i settori corrispondenti a tale fascia (per semplicità il nostro script riempie una sola fascia, ma può essere facilmente esteso per occupare anche le altre). Una volta che le fasce sono occupate, lo script si occupa di effettuare una copia della struttura del file system. Lo scopo di questa operazione è quello di avere a disposizione una versione della Master File Table che indica i settori corrispondenti alle fasce di azzeramento come occupati.

Il passo successivo, da compiere per inizializzare il disco, è svuotarlo completamente dei file fittizi e copiare al suo interno la falsa MFT che lo script ha a disposizione. In questo modo, nonostante le fasce di azzeramento siano vuote, il sistema operativo le considererà occupate e non scriverà al suo interno alcun dato. Questo ci garantisce che qualunque dato visibile dal sistema operativo è recuperabile, e al contempo ci mette a disposizione l'intero spazio delle fasce di azzeramento per occultare dati. Se vogliamo spostare un dato nelle zone nascoste, infatti, lo script si occupa di copiarlo in modalità raw al loro interno, di memorizzare in una struttura dati le informazioni necessarie al recupero (dimensione, posizione dei settori, etc.) e di azzerare la vecchia versione del file visibile da sistema operativo. Nel caso in cui invece si volesse recuperare un dato nascosto, lo script è capace di eseguire le operazioni inverse. Questo processo non è tuttavia limitato ai singoli file: è infatti possibile riempire ogni fascia con un'immagine disco montabile grande come la sua dimensione (4 GB), che ha anche il vantaggio di rimanere sempre della stessa dimensione

semplificando il problema di dover salvare in una struttura dati a parte le informazioni necessarie al recupero dei file.

Una volta che le operazioni di occultamento sono state eseguite, nel caso in cui venga effettuata un'analisi forense del disco gli scenari possibili sono i seguenti:

- Se si è a conoscenza in anticipo di un imminente sequestro del disco, lo script prevede un comando di distruzione dei dati che effettua un backup della MFT corrente, un quick format (il cui effetto sarà solo quello di azzerare i dati nelle fasce) e una nuova copia della MFT che era presente prima della formattazione. Così durante l'analisi è visibile un file system coerente che non desta particolari sospetti, ma in cui i dati nascosti sono stati completamente azzerati.
- Se invece l'utente non ha tempo di lanciare il comando di distruzione, il file system visibile è comunque coerente, ma è possibile rilevare nelle fasce di azzeramento la presenza dei file fittizi che ne occupano l'intero spazio, non recuperabili. Per quanto una simile informazione possa destare sospetti, identificare e recuperare i veri contenuti delle fasce con gli strumenti di analisi tradizionali resta estremamente difficile.

Per concludere, vanno citati alcuni ostacoli a queste operazioni posti da Windows 7 che lo script si occupa di aggirare. Innanzitutto le scritture possono essere effettuate soltanto in multipli della dimensione dei settori, quindi è necessario utilizzare un meccanismo di padding per i dati che si vogliono nascondere. Secondariamente il sistema operativo risponde con degli errori se si prova ad eseguire molte scritture/letture di dati indirizzando direttamente i settori. Per risolvere questo problema, lo script si occupa di smontare e rimontare il disco dopo ogni operazione di I/O, poiché in questo modo il conteggio delle scritture/letture viene azzerato e non si rischia di raggiungere la soglia che genera gli errori.

4.4 Recuperabilità dei file

Gli esperimenti sul disco Corsair ci hanno rivelato che, usando il file system NTFS, esistono zone del dispositivo che non vengono mai azzerate in seguito ad un quick format o a una cancellazione. Abbiamo quindi effettuato degli esperimenti per verificare se i file contenuti in queste zone fossero effettivamente recuperabili.

Come già accennato nel capitolo precedente, abbiamo scelto un file (Figura 4.4) che, laddove presente, fosse sempre recuperabile con il nostro carver (ovvero scalpel). Abbiamo quindi scritto su disco 150000 copie del file, per poi eseguire un quick format.

Attraverso il nostro script il carver è riuscito a recuperarne 101381, di cui 101379 integre, corrispondenti ad una percentuale di recuperabilità del 67.59%. Le immagini non integre inoltre risultavano consecutive, risultato coerente con la presenza di fasce di azzeramento. Abbiamo voluto quindi ripetere l'esperimento con un numero maggiore di file, così da coinvolgere più fasce di azzeramento. Abbiamo scritto su disco 225020 copie dell'immagine, per poi eseguire un quick format. Il risultato è stato coerente con quanto ci aspettavamo: scalpel ha recuperato 174178 jpg di cui 174167 integre, corrispondenti ad una percentuale di recuperabilità del 77.4%. Anche in questo caso le immagini non integre apparivano consecutive, in piccoli gruppi.

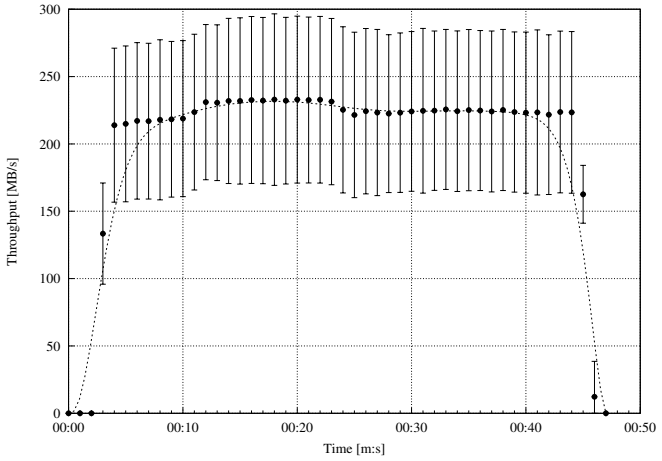
Per quanto riguarda la cancellazione di singoli file, abbiamo usato un approccio simile. Abbiamo azzerato completamente i dischi, li abbiamo riempiti con file diversi dalla nostra immagine in modo da riempire le zone che non ci interessavano e infine abbiamo scritto alcune copie dell'immagine nota in settori interni o esterni alle fasce di azzeramento. Questo è stato possibile usando lo script zero-stripes-finder.py (vedi appendice) per identificare gli indirizzi dei settori delle fasce di azzeramento e hdparm per verificare in che settori si trovassero i file. Abbiamo quindi verificato che cancellando tutte le copie presenti dentro le fasce di azzeramento il carver non era più in grado di recuperarle, mentre cancellando le copie presenti al di fuori di tali fasce la recuperabilità era garantita.

FIGURA 4.4: L'immagine usata per gli esperimenti di carving. La copia originale è scaricabile all'indirizzo: `https://www.dropbox.com/s/kofe0rp7b3gmz4q/flower.jpg`

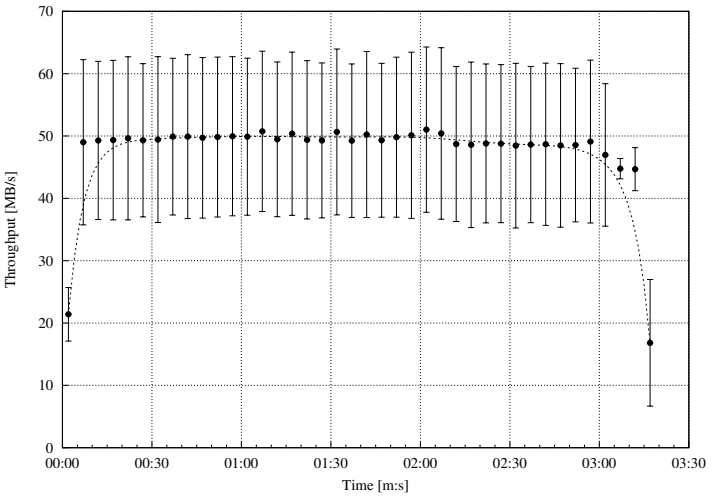
Analogo è stato l'esperimento sul file system ext4, per verificare se il mancato azzeramento di singoli file sul disco Samsung corrispondesse ad una loro effettiva recuperabilità. Abbiamo azzerato completamente il disco, lo abbiamo riempito con file diversi dalla nostra immagine e infine abbiamo scritto un'unica copia dell'immagine nota. Abbiamo quindi cancellato l'immagine e lanciato scalpel per tentare di recuperarla. Il risultato è stato positivo e il carver è riuscito ad estrarre con successo l'immagine dal disco.

4.5 Compressione

Eseguendo lo script che implementa il test sulla compressione su tutti i dischi in nostro possesso abbiamo rilevato una visibile differenza tra le performance dei dischi che comprimono i dati in fase di scrittura e quelle dei dischi che non effettuano alcuna compressione. Un esempio di tale differenza è mostrata in Figura 4.5 relativamente al disco Corsair, dove in presenza di compressione il tempo necessario a scrivere un file ad alta entropia, le cui caratteristiche sono indicate in tabella 4.1, è circa il triplo di quello necessario a scrivere un file a bassa entropia.



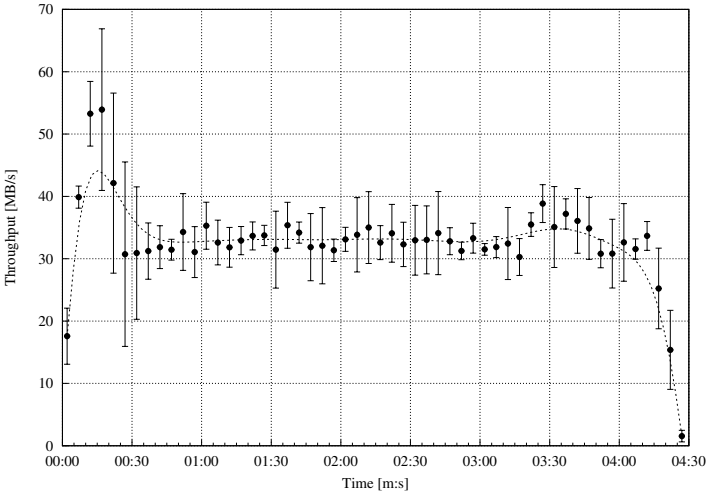
(a) Low entropy



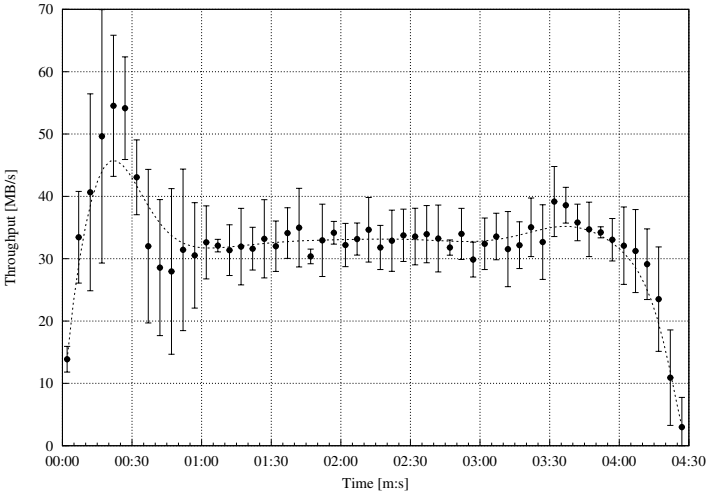
(b) High entropy

FIGURA 4.5: Differenza nelle performance di scrittura di file da 10 Gb nel disco Corsair F60. Ogni punto mostra la media e la varianza dei risultati ottenuti in seguito a 15 esecuzioni dell'esperimento; la linea è un'interpolazione dei valori medi. Nel caso di file a bassa entropia (a) il throughput è considerabilmente maggiore e il tempo necessario al trasferimento è meno di 1/3 di quello che serve per scrivere un file ad alta entropia (b). Questo risultato conferma che una quantità inferiore di dati viene fisicamente scritta su disco, il ché significa che è stata effettuata un'operazione di compressione.

Questa differenza notevole nei trasferimenti conferma la nostra ipotesi che il tempo necessario ad eseguire l'operazione di compressione a livello hardware fosse trascurabile rispetto al tempo di accesso alle memorie. Poichè la compressione avviene in modo completamente trasparente e i dati fisicamente scritti su disco sono notevolmente inferiori, possiamo perciò aspettarci che un'eventuale analisi white-box risulterebbe infruttuosa, restituendo dati pressoché impossibili da decifrare.
Molto diverso è il comportamento del disco Samsung, su cui la funzionalità di compressione è assente. Come mostra la Figura 4.6, il throughput medio (36.12 MB/s) e i tempi di scrittura dei due tipi di file sono pressoché identici, sia nel caso ad alta entropia che in quello a bassa entropia.



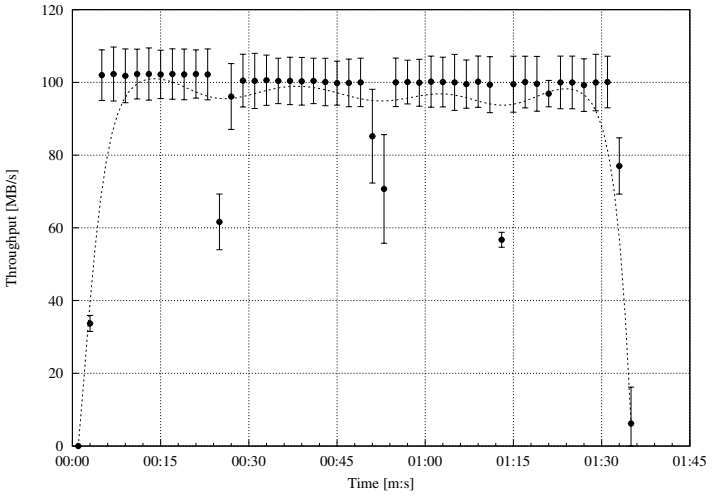
(a) Low entropy



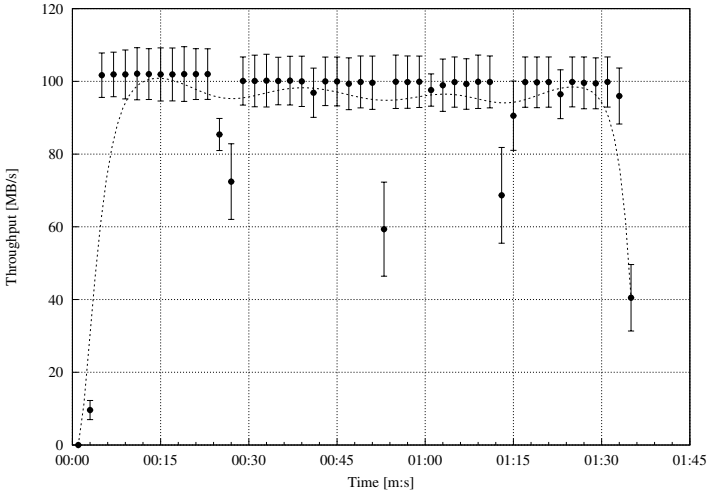
(b) High entropy

FIGURA 4.6: Differenza nelle performance di scrittura di file da 10 Gb nel disco Samsung S470. I grafici relativi ai file a bassa entropia (a) e ad alta entropia (b) hanno circa la medesima forma e durata: ciò mostra che il controller non effettua alcuna operazione di compressione dei dati prima di scriverli in memoria.

Il grafico relativo al disco Crucial, come mostrato in Figura 4.7, seppur con un andamento differente fornisce risultati analoghi a quelli del disco Samsung. Sia le scritture del file ad alta entropia che quelle del file a bassa entropia, infatti, danno risultati pressoché identici sia riguardo al throughput che alla durata. Circa ogni 25 secondi è inoltre possibile notare dei cali di performance che si ripetono ad ogni esecuzione del test e non sono riscontrabili in condizioni analoghe negli altri dispositivi. Per indagarne la causa abbiamo ripetuto gli esperimenti sotto condizioni differenti (ad esempio variando dimensione dei file, sistema operativo o computer). I cali si sono tuttavia ripresentati in qualunque contesto; la nostra ipotesi è che siano causati da computazioni interne effettuate dal controller ad intervalli regolari.



(a) Low entropy



(b) High entropy

FIGURA 4.7: Differenza nelle performance di scrittura di file da 10 Gb nel disco Crucial M4. Come in Figura 4.6, entrambi i trasferimenti hanno circa la stessa forma e durata. Si noti inoltre che il throughput è considerevolmente maggiore rispetto agli altri dispositivi.

File	Rapporto di compressione gzip	7zip	bz2	Entropia
/dev/zero	1,030.42	7,086.90	1,420,763.25	0.0
/dev/urandom	1.0	0.99	0.99	0.99

TABELLA 4.1: File utilizzati per i test di compressione: i file con valori di entropia più alti sono difficili da comprimere e quindi risultano in una maggiore quantità di dati da scrivere su disco. Per ogni file è riportato il rapporto tra la dimensione iniziale del file e la dimensione del file compresso utilizzando tre noti algoritmi di compressione.

Nel grafico in Figura 4.6 è infine importante notare la presenza di un transitorio iniziale con un throughput visibilmente maggiore rispetto alla media riscontrata nel dispositivo. L'aspetto interessante di questi transitori è che la loro dimensione è paragonabile alla grandezza della cache del disco, nonostante sia stato inviato un comando di disabilitazione tramite hdparm. Tale comando tuttavia funziona correttamente con i dischi Corsair e Crucial, poichè come è possibile vedere nelle figure 4.5 e 4.7 negli esperimenti effettuati con tali dischi non è presente alcun transitorio. La nostra ipotesi è che la causa di tale comportamento vada ricercata nel controller, il quale comunica al sistema operativo di aver disabilitato la write cache ma in realtà maschera il comando ricevuto e continua ad utilizzarla. Ciò non influisce in ogni caso sulla validità dei nostri esperimenti, poichè abbiamo scelto di utilizzare file di dimensioni molto maggiori rispetto alla capacità di tale cache e quindi, una volta che essa viene saturata, è possibile osservare il comportamento che ci interessa ai fini del test.

4.6 Wear leveling

Abbiamo eseguito i test black box per l'analisi della funzionalità di wear leveling su tutti i dischi in nostro possesso. Come pattern noto abbiamo creato un file di testo di 2MB contenente la parola *PROVA* ripetuta per tutta la lunghezza del documento. Come carver in questo caso abbiamo scelto photorec, poichè nei nostri esperimenti si è rivelato il più affidabile nel recupero di file .txt. Abbiamo riempito i dischi alle varie percentuali proposte e quindi abbiamo cominciato le sovrascritture multiple del file noto.

Nonostante la nostra scelta del valore di sovrascritture (10000) si avvicinasse molto al valore di usura delle celle dichiarato dai produttori, in nessun caso siamo riusciti a rilevare un cambiamento visibile dello stato del disco. L'analisi in tempo reale dello spazio libero non ha mai mostrato alcuna variazione dovuta a presenza di copie multiple del file, il carver ha sempre estratto una sola copia del nostro file noto e l'analisi dei settori effettuata con hdparm ha sempre dato risultati identici sia ad inizio che a fine esperimento. La nostra ipotesi è che questa assenza di risultati sia dovuta ad un mascheramento effettuato dal controller. Molti dei nostri dischi infatti possiedono un certo livello di

overprovisioning (ad esempio il disco Corsair ha 64 GB fisici contro 60 GB nominali). É molto probabile che gli indirizzi logici esposti dal controller siano rimasti costanti, mentre gli indirizzi fisici ad essi associati siano stati modificati in modo trasparente al sistema operativo. É quindi possibile che eventuali copie multiple del file noto si trovino in zone della memoria non accessibili con un'analisi black box, poichè il controller espone solo la copia più recente. Per effettuare un'analisi delle funzionalità di wear leveling sui nostri dischi sarebbe quindi necessario adottare un approccio white-box.

4.7 Write amplification

Abbiamo eseguito l'analisi degli attributi S.M.A.R.T. come ultimo test della serie di esperimenti, così da ottenere dei valori influenzati dall'attivazione di tutte le funzionalità precedentemente analizzate. Per quanto riguarda il carico di lavoro generato dal nostro script, abbiamo notato che sono necessarie almeno due ore di scritture continuate per poter rilevare cambiamenti significativi nei valori esposti. I risultati del test sul disco Corsair indicano che ad un totale di 512 GB di scritture logiche corrispondono 236 GB scritture fisiche, per una write amplification pari a 0.4625. Il risultato è coerente con la presenza di compressione sul disco rilevata in 4.5, il cui effetto è talmente aggressivo da occultare gli eventuali aumenti dei valori degli attributi S.M.A.R.T. legati a wear leveling e scritture non sequenziali.
Non è stato invece possibile testare i dischi Crucial e Samsung, in quanto non espongono gli attributi S.M.A.R.T. necessari.

4.8 Classificazione black box dei dischi testati

Dall'esito degli esperimenti black box illustrati possiamo classificare i dischi testati secondo due metriche: analizzabilità black box e utilizzabilità in ambito processuale.

Per quanto riguarda l'analizzabilità black box, le classi sono le seguenti:

- Corsair F60: Sotto NTFS la classe da associare al disco è B, in quanto grazie alle fasce fisse di azzeramento è garantita recuperabilità parziale sia in seguito a cancellazione di singoli file che in seguito a quick format. Sotto ext4 invece la classe è C, poichè i dati sono stati azzerati in pochi secondi nel corso di ogni test effettuato.
- Crucial M4: Sia con NTFS che con ext4 la classe da associare a questo disco è C, poiché i dati vengono sempre azzerati nell'arco di pochi secondi.

- Samsung S470: Sotto NTFS il disco va classificato come di tipo C, poichè tutti i test effettuati hanno rilevato un azzeramento totale dei dati entro pochi secondi. Sotto ext4 il disco è invece di classe C nel caso di quick format, ma di classe A nel caso di cancellazione di singoli file attraverso il comando del sistema operativo.

Per quanto riguarda l'integrità dei dati, infine, l'hash è risultato stabile in tutti gli esperimenti, quindi la classe da associare a tutti e tre i dischi è +. In tabella 4.2 riportiamo tale classificazione solo per i dischi che garantiscono una recuperabilità dei dati almeno parziale, in quanto non ha significato nel caso in cui tutto il contenuto del disco venga azzerato.

SSD	**FS**	**TRIM**	**GC**	**Recuperabilità (quick format)**	**Recuperabilità (cancellazione)**	**Classe (format | canc.)**
Corsair F60	**NTFS**	Parziale	×	Parziale	Parziale	**B+ | B+**
	ext4	Completo	×	Nulla	Nulla	**C | C**
Crucial M4	**NTFS**	Completo	×	Nulla	Nulla	**C | C**
	ext4	Completo	×	Nulla	Nulla	**C | C**
Samsung S470	**NTFS**	Completo	×	Nulla	Nulla	**C | C**
	ext4	Completo	×	Nulla	Completa	**C | A+**

Tabella 4.2: Risultati dei test per i dischi in nostro possesso e relativa classificazione.

5. Conclusioni

Nel corso di questo lavoro abbiamo cercato di fornire un nostro contributo alla soluzione del problema fondamentale alla base dell'analisi forense di memorie a stato solido, ovvero scoprire se le comuni tecniche di acquisizione e analisi di dischi magnetici possano essere applicate a questa nuova tecnologia. I nostri risultati hanno dimostrato che i due tipi di dispositivi non possono essere trattati allo stesso modo, in quanto gli strumenti attualmente utilizzati per l'analisi forense di memorie di massa si basano sul presupposto che il disco non alteri autonomamente i dati e che quindi tutti i dati non sovrascritti siano recuperabili. I controller dei dischi a stato solido, invece, oltre a mascherare il contenuto delle memorie sono in grado di eseguire operazioni di azzeramento dei blocchi, compressione o cifratura dei dati senza l'intervento del sistema operativo; alimentare il dispositivo è condizione sufficiente per l'avvio di operazioni capaci di compromettere la recuperabilità dei dati. Non esiste inoltre alcuno standard per l'implementazione di tali funzionalità, che possono cambiare radicalmente da produttore a produttore. Per questo motivo abbiamo scelto di definire ed implementare una procedura di test per guidare l'analisi di questo tipo di dispositivi, con lo scopo di evitare la propagazione degli errori attualmente presenti nei lavori di ricerca in questo ambito. Abbiamo inoltre proposto diverse metriche per classificare le varie dimensioni di analizzabilità dei dischi a stato solido.

L'esecuzione dei test sui dispositivi in nostro possesso ha mostrato che, nonostante i mascheramenti effettuati dal controller ci abbiano permesso di analizzare solo un sottoinsieme delle funzionalità teoricamente presenti, tali funzionalità sono risultate sufficienti per compromettere la recuperabilità dei dati cancellati nell'arco di pochi secondi. Al momento le problematiche sono parzialmente mitigate dal fatto che soltanto pochi sistemi operativi recenti supportano di default le funzionalità più distruttive (ad esempio il TRIM), ma dobbiamo aspettarci che con la crescente diffusione dei dischi a stato solido molti altri futuri sistemi operativi adotteranno il medesimo approccio. Inoltre anche a livello di controller, a causa della crescente richiesta di prestazioni, si stanno sempre più diffondendo funzionalità (come la compressione) che possono ostacolare le metodologie di recupero dei dati white box che cercano di aggirare i problemi introdotti dai controller.

Per quanto i nostri esperimenti abbiano rilevato che uno dei controller più diffusi garantisce la recuperabilità di buona parte del contenuto del disco, dato apparentemente positivo, questo mostra anche che l'analizzabilità forense dei dischi a stato solido dipende completamente dalle scelte implementative (o dagli errori) dei singoli produttori. Essi tuttavia non hanno alcun interesse a preservare l'analizzabilità dei loro dischi, poiché significherebbe rinunciare a molte funzionalità di ottimizzazione dei loro dispositivi, portando a un calo delle performance e rendendoli meno competitivi sul mercato. Per questo motivo ci aspettiamo che situazioni come quella rilevata siano solamente eccezioni, destinate a diventare sempre più rare. In ogni caso abbiamo dimostrato che, anche laddove presente, la possibilità di recuperare parte dei dati può fornire ad un utente esperto i mezzi per occultare e rendere irrecuperabili dati a proprio piacimento.

Dal punto di vista della analisi white box, i numerosi tentativi effettuati senza successo ci portano a credere che l'accesso ai chip NAND in modo semplice e a basso costo non sia ancora possibile a causa della eterogenità dei chip, della mancanza di documentazione, del costo elevato degli strumenti professionali e della opposizione dei produttori che non forniscono modalità di debug documentate nè evaluation board. Abbiamo inoltre provato a contattare direttamente alcuni di loro per richiedere se fosse disponibile materiale aggiuntivo oltre a quanto pubblicamente disponibile, ma essi si sono dimostrati poco collaborativi, certamente con lo scopo di proteggere tecnologie proprietarie sviluppate internamente.

Per quanto riguarda gli sviluppi futuri di questo lavoro, uno dei nostri limiti principali è stato il ridotto numero di dispositivi a nostra disposizione. Per poter generalizzare i risultati ottenuti, infatti, sarebbe necessario effettuare i test proposti su un'ampia gamma di dischi a stato solido con funzionalità eterogenee. Idealmente, sarebbe desiderabile trovare almeno un dispositivo i cui comportamenti possano essere associati ad ogni classe. Inoltre una funzionalità fondamentale che non siamo riusciti a rilevare e studiare è stata quella di garbage collection. Essa è legata ad uno dei principali problemi teorici dell'analisi forense di memorie SSD, ovvero la stabilità dell'hash. Uno degli sviluppi futuri più urgenti, quindi, sarebbe proprio effettuare i test su un dispositivo dotato di garbage collector implementato a livello di controller, così da studiarne l'aggressività. Sarebbe poi interessante studiare se è possibile definire una metodologia di acquisizione alternativa in caso di instabilità dell'hash, ad esempio rilevando tempi e condizioni di attivazione del garbage collector per vari controller ed effettuando acquisizioni di immagini (o calcoli di hash) parziali che vadano combinati insieme, togliendo periodicamente l'alimentazione al disco prima che la funzionalità di garbage collection si attivi.
Un altro percorso che potrebbe essere interessante riguarda l'analisi degli algoritmi di compressione e cifratura implementati all'interno dei controller. In particolare, potrebbe essere utile cercare un modo per recuperare la chiave di cifratura interna al disco oppure

studiare il funzionamento dell'algoritmo di compressione per decomprimere i dati recuperati con metodi white box. Infine per rendere possibile uno studio approfondito della recuperabilità dei dati con approcci di tipo white box sarebbe fondamentale la progettazione e la realizzazione di un hardware a basso costo capace di interfacciarsi in modo semplice e stabile alle memorie NAND di uso più comune, aprendo così grandi possibilità di studio dei controller e consentendo ad ogni analista forense nuove possibilità di analisi senza la necessità di creare complessi e costosi hardware.

Appendice A

I tool di analisi sviluppati

Questa appendice contiene il codice dei tool che abbiamo sviluppato per l'esecuzione automatica dei vari test. Il codice delle implementazioni dei test può essere invece trovato nell'appendice B.

A.1 Script antiforense per controller Sandforce (sandforce-antiforensics.py)

Prima di eseguire questo script è necessario collegare al computer il disco su cui si vuole operare, altrimenti l'esecuzione viene terminata istantaneamente. Una volta lanciato, lo script presenta un menu interattivo con le operazioni che possono essere eseguite.

```
#!/usr/bin/python
#-*- coding: latin-1 -*-
import os
import sys
import random
import cPickle
import time
import gzip
import base64
import string
import StringIO
#import cStringIO

#dev.flush()
#os.fsync(fd)
BUF_SIZE = 2048000
WRITE_CHUNK_SIZE = 1024000
MBR_POSITION = 0
MFT_POSITION = 3222274048
MFT_MIRR_POSITION = 1048576
MFT_POSITION_SECTOR = 6293504
```

```
MFT_MIRR_POSITION_SECTOR = 2048
LENGTH_MBR = 512
LENGTH_MFT = 5120000
LENGTH_MFT_MIRR = 1024000

#check if user is root
#euid = os.geteuid()
#if euid != 0:
#    print "Script not started as root.... exit"
#    exit(0)
#c = wmi.WMI()
#for physical_disk in c.Win32_DiskDrive(Index=1,InterfaceType='USB'):
#    sector_size = int(physical_disk.BytesPerSector)
#    total_sectors = int(physical_disk.TotalSectors)
#    print total_sectors

stripes = ((10000,20000),(20000000000,25000000000) ,(80000,90000))

class HiddenFile:
    def __init__(self, path, hidden_begin, size, padding):
        self.path = path
        self.hidden_begin = hidden_begin
        self.size = size
        self.padding = padding
    def get_size(self):
        return self.size

class DiskStructure:
    def __init__(self, mbr, mft, mftmirr):
        self.mbr = mbr
        self.mft = mft
        self.mftmirr = mftmirr

file_list = []
diskstruct = DiskStructure("","","")


def get_file_list():
    print ' ------------------------------------------------------------'
    print '|Name                          Position            Size      |'
    for f in file_list:
        print '|{0:<30}{1:<20}{2:<10}|'.format(f.path, f.hidden_begin, f.↘
         → get_size())

    print ' ------------------------------------------------------------'

def add_file():
    filepath = raw_input('Enter file path:')
    #copio il file nella zona trimmata
    #sovrascrivo con random il vecchio file e poi lo elimino con rm
    f_info = os.stat(filepath)
    padding = 0
    #tengo spazio per padding se serve
    if not f_info.st_size % 512 == 0:
        padding = ((( f_info.st_size / 512 ) + 1 ) * 512) - f_info.st_size
```

```
    #i file si concatenano uno dopo l'altro senza spazio libero quindi cerco l' ↘
        → ulimo elemento della lista e mi sposto
    #una posizione oltre la sua grandezza
    if len(file_list) == 0:
        free_position = stripes[1][0]
    else:
        #le dimensioni di tutti i file devono essere multipli di 512
        free_position = file_list[-1].hidden_begin + file_list[-1].size + ↘
        → file_list[-1].padding + 1
    #when exiting the safe zone, the operation is cancelled (file dimension is ↘
        → rounded with multiples of 512 to avoid write problems)
    if (free_position + f_info.st_size) > stripes[1][1]:
        print "No free space available",free_position + f_info.st_size
        return 0
    file_list.append(HiddenFile(filepath, free_position, f_info.st_size, ↘
        → padding))
    print "File located at: %s added" % filepath

def remove_file():
    i = 0
    for file in file_list:
        print '{}- {}'.format(i, file.path)
        i += 1

    index = raw_input('Chose the file to delete:')
    #f = open(file_list[ind].path, "w")
    #dev.seek(file.hidden_begin)
    #f.write(dev.read)
    #f.close()
    elem = file_list.pop(int(index))
    print "File located at: %s removed" % elem.path

def init_disk():
    #prendere una immagine iniziale (magari compressa) con dentro mbr, mft ecc
    #2 scrivo sul disco nei punti giusti
    #print "Current I/O pointer position :%d" % dev.tell()
    #ricordarsi che si possono scrivere solo multipli di 512

    with gzip.open('diskstruct-init.dat', 'rb') as data:
        disk = cPickle.load(data)

    #fd_local = os.dup(fd)
    #dev = os.fdopen(fd_local, "wb",1024)
    #fd = os.open("\\\\.\\PHYSICALDRIVE1", os.O_RDWR|os.O_BINARY)
    #dev = os.fdopen(fd, "rb")
    #dev.seek(30000000000)

    #questo funziona
    #for i in xrange (512, 51200, 512):
        fd = os.open("\\\\.\\PHYSICALDRIVE1", os.O_WRONLY|os.O_BINARY,BUF_SIZE)
        os.lseek(fd, 0 , 0)
        os.lseek(fd, 1024000 , 1)
        os.lseek(fd, 1024000 , 1)
        os.write(fd, b'\x88' * 512)
```

```
        os.close(fd)
    #   print "giro", i

    write_raw(MBR_POSITION, disk.mbr)

    #to solve the error caused by too large integers, I pass as arguments the ↘
        → beginning and the relative dimension
    #print MFT_POSITION + LENGTH_MFT
    #for i in xrange (0, len(disk.mft), WRITE_CHUNK_SIZE):
    #   write_raw(MFT_POSITION + i, disk.mft[i : i + WRITE_CHUNK_SIZE])

    #for i in xrange (0, LENGTH_MFT_MIRR, WRITE_CHUNK_SIZE):
    #    write_raw(MFT_MIRR_POSITION + i, disk.mftmirr[i : i + WRITE_CHUNK_SIZE ↘
        → ])

    diskstruct = disk
    #3 do messaggio di ok
    print "Disk initialization complete"

def write_raw(position,data):
    fd = os.open("\\\\.\\PHYSICALDRIVE1", os.O_WRONLY|os.O_BINARY, BUF_SIZE)
    os.lseek(fd, position, 0)
    os.write(fd, data)
    os.close(fd)

def read_raw(position, dim):
    fd = os.open("\\\\.\\PHYSICALDRIVE1", os.O_RDONLY|os.O_BINARY, BUF_SIZE)
    os.lseek(fd, position, 0)
    data = os.read(fd, dim)
    os.close(fd)
    return data

def restore_all():
    #per ora non tengo conto di file che possono cambiare dimensione
    for file in file_list:
        f = open( file.path, "wb" )
        dev.seek( file.hidden_begin )
        dev.read( file.size + file.padding )
        #os.lseek(fd, file.hidden_begin, 0)
        #devo leggere per forza multipli di 512
        #temp = os.read(fd, file.size + file.padding)
        #scrivo solo
        f.write( temp[ 0 : file.size ] )
        f.close()
    print "File restore complete"
    pass

def hide_all():
    #1 copy the content of the file in the hidden zone (size is fixed because ↘
        → it's a disk image)
    #2 wipe and delete the visible file

    # da sistemare: se il file si modifica in grandezza devo aggiornare la ↘
        → entry nella mia struttura dati o si sballa tutto
    # considero solo immagine disco che non cambia in grandezza
```

```
    for file in file_list:
        f = open(file.path, "rb+")
    #questo deve essere rifatto con yeld per essere lazy
        content = f.read() +  b'\x00' * int(file.padding)
        print '####', len(content)
        for i in xrange (0, file.size, WRITE_CHUNK_SIZE):
            write_raw(file.hidden_begin + i, content[i : i + WRITE_CHUNK_SIZE])
    #real file sanitization and delete
        f.seek(0)
        #for i in xrange(0,file.size):
            #f.write(chr(random.randint(33,126)))
        f.write(b'\x00' * file.size)
        f.close()
        f.flush()
        os.remove(file.path)
        print "Sanitizzato ed eliminato file %s" % file.path

def destroy():
    #faccio quick format e restore prendendo dati da diskstruct
    print "I'm in destroy"
    pass

def quit():
    with gzip.open('file_list.dat', 'wb') as data:
        cPickle.dump(file_list, data)
    #try:
    #    dev.close()
    #except IOError:
    #    print "Errore nella chiusura del file dev"
    #try:
    #    os.close(fd)
    #except OSError:
    #     print "Errore nella chiusura del file descriptor"
    exit(0)

def resume():
    #da aggiungere try except, se non trovo il
    #file creo un nuovo file_list globale vuoto
    global file_list
    try:
        with gzip.open('file_list.dat', 'rb') as data:
            file_list = cPickle.load(data)
    except IOError as e:
        pass

    diskstruct.mbr = read_raw(MBR_POSITION, LENGTH_MBR)
    diskstruct.mft = read_raw(MFT_POSITION, 2048000)
    diskstruct.mftmirr = read_raw(MFT_MIRR_POSITION, LENGTH_MFT_MIRR)

    #os.lseek(fd, MBR_POSITION, 0)
    #diskstruct.mbr = os.read(fd, LENGTH_MBR)
    #os.lseek(fd, MFT_POSITION, 0)
    #diskstruct.mft = os.read(fd, 2048000)
    #os.lseek(fd, MFT_MIRR_POSITION, 0)
    #diskstruct.mftmirr = os.read(fd, LENGTH_MFT_MIRR)
```

```
    save_on_file()
    print "Disk structure updated"

def save_on_file():
    with gzip.open('diskstruct.dat', 'wb') as data:
        cPickle.dump(diskstruct, data)

def print_menu():
    menu = """
    (i) - initialize the disk
    (l) - get the hidden file list
    (a) - add a file
    (r) - remove a file
    (e) - restore all the files
    (h) - hide all the files
    (d) - destroy everything
    (q) - quit the program
    """
    print menu

actions = {"i": init_disk, "l": get_file_list, "a": add_file, "r": remove_file, ↘
        → "e": restore_all, "h": hide_all, "d": destroy, "q": quit}

resume()
while True:
    print_menu()
    selection = raw_input("Your selection: ")
    if not(selection):
        continue
    try:
        toDo = actions.get(selection)
    except:
        continue
    toDo()
```

A.2 Carving e controllo di integrità (carver.sh)

Prima di lanciare lo script, è importante modificare le prime due righe del file inserendo rispettivamente il path del file scalpel.conf e il path di un documento di testo contenente gli hash dei file usati per l'esperimento.
Per eseguire il carving di un'immagine disco chiamata DISKIMAGE.dd, lanciare il comando *sudo ./carver.sh DISKIMAGE.dd*

```
#!/bin/bash

SCALPEL_CONF_PATH=scalpel.conf
HASH_FILE_PATH=hashlist.txt
TOTAL_COUNT=0
RECOVERABLE_COUNT=0

#The image is analyzed by Scalpel and the output is
#stored in the subdirectory scalpel-output automatically
#created in the directory where the script is launched.
scalpel -v -c $SCALPEL_CONF_PATH $1
chmod -R 777 scalpel-output/

#For each file contained in the directory scalpel-output
#the hash is calculated and compared with known hashes.
dirlist=$(find scalpel-output/ -mindepth 1 -type d)
for dir in $dirlist; do
    echo -e "##Analyzing directory $dir..." | tee -a $LOGFILE; date >> $LOGFILE
    files=$(find $dir -mindepth 1)
    for filename in $files; do
        filehash=$(md5sum $filename | cut -d ' ' -f 1)
        if grep -Fxq "$filehash" $HASH_FILE_PATH
        then
            RECOVERABLE_COUNT=$(($RECOVERABLE_COUNT+1))
        fi
            TOTAL_COUNT=$(($TOTAL_COUNT+1))
        done
done

#Printing output
PERCENT=`echo "scale=2; (($RECOVERABLE_COUNT*100)/$TOTAL_COUNT)" |bc`
echo "##Number of analyzed files: $TOTAL_COUNT" | tee -a $LOGFILE; date >> ↘
        → $LOGFILE
echo "##Number of recoverable files: $RECOVERABLE_COUNT" | tee -a $LOGFILE; ↘
        → date >> $LOGFILE
echo -ne "##Percentage of recoverable files: $PERCENT \n"| tee -a $LOGFILE; ↘
        → date >> $LOGFILE
```

A.3 Analisi in tempo reale (comparer.sh)

Questo script effettua una comparazione byte a byte dei campioni prelevati da due device dati in input. Per seguirlo lanciare *sudo comparer.sh DEVICEA DEVICEB*. In questo lavoro lo script è stato usato ponendo uno dei due device come /dev/zero, così da misurare lo spazio libero rimanente su disco.
Prima di eseguire lo script, l'utente deve modificare la variabile *size* inserendo la dimensione del suo disco in MB. Anche le variabili *samplesize* e *gap* possono essere modificate a piacere: aumentando la prima e/o riducendo la seconda si ottiene una maggiore precisione a costo di un tempo di esecuzione più lungo.
L'idea dello script è stata presa da [10] e liberamente adattata.

```
#!/bin/bash

SAMPLESIZE=10000; # bytes to sample
SIZE=60000; # SIZE of disk, megabytes. PLEASE REMEMBER TO SET THIS.
GAP=1000; # GAP between samples in megabytes.
LOGFILE=comparer.log

date >> $LOGFILE; echo -e "Starting comparer..." > $LOGFILE;
while (true)
do
  total_diff=0;
  percent=0;
  diff_sample_sum=0;
  total_samples=0;

  for (( P=0; P<SIZE; P=P+GAP )); do #
        #compare 2 streams bitwise and write number of differences
        newsample=`cmp -l $1 $2 -n $SAMPLESIZE -i"$P"MB | wc -l | bc`
        diff_sample_sum=$((newsample+diff_sample_sum))
        #count the total number of compared bytes
        total_samples=$((total_samples+SAMPLESIZE))
  done

  date >> $LOGFILE; echo -ne "$total_samples B compared, $diff_sample_sum are ↘
        → different    " >> $LOGFILE

  total_diff=`echo $diff_sample_sum | awk 'BEGIN{t=0}{t += $1}END{print t}'`
  percent=`echo "scale=2; 100-(($total_diff*100)/$total_samples)" |bc`
  echo -ne "Streams are equal at $percent%\n\n" >>  $LOGFILE
  sleep 1
done
```

A.4 Misuratore di entropia (entro-py.py)

Per misurare l'entropia di un file FILEPATH, lanciare il comando *python entro-py.py FILEPATH*

```
import sys
import math
import os

f1 = open(sys.argv[1], "rb")
file_size = os.path.getsize(sys.argv[1])
byte_counts = [0]*256
entropy = 0

#increment values of byte_counts array
while True:
    try:
        chunk = f1.read(4096)
        for byte in chunk:
            byte_counts[ord(byte)] += 1
        #shows progress (not necessary)
        index = f1.tell()
        if index % (2000*4096) == 0:
            print index, "bytes analyzed"
    except:
        break

#Calculate shannon's entropy
print file_size
for count in byte_counts:
    # If no bytes of this value were seen in the value, it doesn't affect the ↘
     → entropy of the file.
    if count == 0:
        continue
    #p is the probability of seeing this byte in the file, as a floating-point ↘
     → number
    p = 1.0 * count / file_size
    entropy -= p * math.log(p, 256)
    print "entropy of", count, "calculated"

f1.close()

print "The entropy of this file is:", entropy
```

A.5 Riempimento dischi (filler.py)

Questo script si occupa di automatizzare il riempimento dei dischi. Per eseguirlo sul disco DISKPATH, lanciare il comando *python filler.py DISKPATH FILESIZE DATAAMOUNT*, dove FILESIZE è la dimensione del file espressa attraverso una delle costanti definite nello script (da SIZE_XS a SIZE_XXL) mentre DATAAMOUNT è la quantità di dati che si vuole scrivere, espressa in GB.

```
import os, sys, random, hashlib

euid = os.geteuid()
if euid != 0:
    print "script not started as root... \nexit..."
    exit(0)

SIZE_XS = 1000 #1KB
SIZE_S = 100000 #100KB
SIZE_M = 1000000 #1MB
SIZE_L = 10000000 #10MB
SIZE_XL = 100000000 #100MB
SIZE_XXL = 1000000000 #1GB

FILL_AMOUNT = 32 #GB
FILE_SIZE =  SIZE_M

if len( sys.argv ) < 2 or len(sys.argv) > 4:
    print "wrong number of parameters... \nexit..."
    exit(0)

if len( sys.argv ) == 2:
    print "file size missing, setting to default value (%s) Bytes..."%FILE_SIZE
    print "data amount missing, setting to default value (%s)... GB"% ↘
        → FILL_AMOUNT

if len( sys.argv ) == 3:
    FILE_SIZE = sys.argv[2]
    if( int( FILE_SIZE ) > SIZE_XXL or int( FILE_SIZE ) <= 0 ):
        print "file size error...\nexit..."
        exit(0)
    print "setting file size to %s Bytes..."%FILE_SIZE
    print "data amount missing, setting to default value (%s) GB..."% ↘
        → FILL_AMOUNT

if len( sys.argv ) == 4:
    FILE_SIZE = sys.argv[2]
    FILL_AMOUNT = sys.argv[3]
    if( int( FILE_SIZE ) > SIZE_XXL or int( FILE_SIZE ) <= 0 ):
        print "file size error...\nexit..."
        exit(0)
    if( int( FILL_AMOUNT ) > 1000 or int( FILL_AMOUNT ) <= 0 ):
        print "fill amount error...\nexit..."
        exit(0)
```

```
    print "setting file size to %s Bytes..."%FILE_SIZE
    print "setting data amount to %s GB..."%FILL_AMOUNT

PATH = sys.argv[1]
FILL_AMOUNT = int( FILL_AMOUNT )
FILE_SIZE = int( FILE_SIZE )
ITERATIONS = (FILL_AMOUNT * 1000000000) / FILE_SIZE
MEMFILE = os.urandom(FILE_SIZE)

print 'md5 hash of the file:  %s' % hashlib.md5(MEMFILE).hexdigest()
print 'sha1 hash of the file:  %s' % hashlib.sha1(MEMFILE).hexdigest()
print "creating %d files..." %ITERATIONS

for i in xrange(0, ITERATIONS):
    with open( "%s/%d" % (PATH,i) , "wb" ) as out_file:
        out_file.write(MEMFILE)
        sys.stdout.write("\rFile %d created..." %(i+1))
        sys.stdout.flush()
```

A.6 Visualizzatore differenze tra immagini disco (visualizer.py)

Questo script produce una rappresentazione grafica delle differenze, settore per settore, tra due immagini disco. Ogni pixel è un settore. I pixel blu indicano settori immutati, i pixel verdi settori azzerati, i pixel rossi settori che hanno subito diverse forme di modifica. Il formato dell'output è ppm, poiché permette di ingrandire l'immagine a piacere per analizzare dettagliatamente ogni settore. I file prodotti sono di grandi dimensioni (circa 1.3 MB per ogni GB del disco), ma una volta compressi occupano pochi KB.
Per eseguire lo script su due immagini IMG1.dd e IMG2.dd, lanciare il comando *python visualizer.py IMG1.dd IMG2.dd OUTPUT.ppm*, dove OUTPUT.ppm è il path del file di output desiderato.

```
import sys, operator, os, time
import numpy as np
from numpy import dtype

#disk_img_dim = 90
sector_size = 4096
f1 = open(sys.argv[1], "rb")
f2 = open(sys.argv[2], "rb")
out_file = open(sys.argv[3],"w")

#Determine the number of sectors through disk image sizes

file1_size = os.path.getsize(sys.argv[1])
file2_size = os.path.getsize(sys.argv[2])
num_sector = max(file1_size,file2_size)/sector_size

#Used For the progress bar
percent = 0
#Set dimensions of the output ppm image to the square root of the number of ↘
        → sectors
output_img_width = int(num_sector**(1.0/2))
output_img_height = int(num_sector**(1.0/2))

#Or set it manually
#output_img_width = 1000
#output_img_height = 600
t=time.time()
out_file.write("P3\n%s %s\n1\n" %(output_img_width,output_img_height))
try:
   for i in xrange(0,num_sector):
      if i%(num_sector/100) == 0:
         #Faster version
         #print percent, "%"
         #Good looking but slower version of percentage print
         sys.stdout.write("\r%d%%" %percent)
         sys.stdout.flush()
```

```
            percent += 1
        #Read a sector and convert it to the correct format
        sector1 = f1.read(sector_size)
        sector2 = f2.read(sector_size)
        conv_sec1 = np.fromstring(sector1, dtype=np.uint64)
        conv_sec2 = np.fromstring(sector2, dtype=np.uint64)

        #Do a Xor between Sectors using numpy
        xor = np.bitwise_xor(conv_sec1, conv_sec2)
        #To do a faster Xor decomment the following line and comment conv_sec1 ↘
           → line (not necessary because the disk is the bottleneck)
        #xor = np.bitwise_xor(np.frombuffer(sector1, dtype=np.uint64), conv_sec2, ↘
           →  conv_sec2)

        #If xor has a byte that is not zero, sectors are different
        if np.any(xor != 0):
            #if the sector form the second image has a non zero byte, the sector ↘
            → has changed but it hasn't been zeroed
            if np.any(conv_sec2 != 0):
                out_file.write("1 0 0 ")
            #if all the bytes are zero, the sector has been zeroed
            else:
                out_file.write("0 1 0 ")
        #if xor is zero, two sectors are identical
        else:
            out_file.write("0 0 1 ")

finally:
        f1.close()
        f2.close()
        out_file.close

#print the total execution time
print "\n", time.time()-t
```

A.7 Generatore di scritture per test su attributi S.M.A.R.T. (workload_generator.py)

Questo script ha l'obbiettivo di generare il carico di lavoro necessario a causare alterazioni visibili dei valori degli attributi S.M.A.R.T. dei dischi. Per eseguire lo script su un device DEV, lanciare il comando *python workload_generator.py DEV TEST_TIME*, dove TEST_TIME è il tempo di esecuzione dello script espresso in secondi. E' inoltre importante modificare la costante DEVICE_SIZE dello script con la dimensione in GB del proprio disco.

```
import os,sys,time

DEVICE = sys.argv[1]
DEVICE_SIZE = 60 * 1000000000 #GB
CHUNK_SIZE = 100000000
WORKLOAD = int( sys.argv[2] )
TIME = int( sys.argv[3] )
MEMFILE = (b'\x00' * CHUNK_SIZE)

print "## workload generator started..."
start_time = time.time()
while (time.time() - start_time) < TIME:
    print "## workload in progress..."

    with open( DEVICE , "wb" ) as dev:
        dev.seek(0)
        for i in xrange(0, DEVICE_SIZE, CHUNK_SIZE):
            if( i % 100000000 == 0):
                countdown = TIME - (time.time() - start_time)
                if countdown < 0:
                    break
                else:
                    sys.stdout.write("\r## %ds left..." %(countdown))
                    sys.stdout.flush()
            dev.write(MEMFILE)
            dev.flush()
    print ""

print "## workload generator stopped..."
```

A.8 Misuratore del tempo di scrittura (write-benchmark.py)

Lo scopo di questo script è quello di effettuare la scrittura di una grande quantità di dati raw ad alta o bassa entropia sul device passato come parametro, per poi misurare il tempo impiegato. Il numero di GB da scrivere può essere modificato cambiando il valore di una variabile globale interna allo script.
Per eseguire lo script su un device DEV, lanciare il comando *python write-benchmark.py DEV ENTROPY_PARAM*, dove ENTROPY_PARAM può assumere il valore *high* se si desidera scrivere un flusso di dati ad alta entropia oppure *low* se si desidera scrivere un flusso di dati a bassa entropia.

```
import sys,time,os

DATA_AMOUNT_GB = 10
DEVICE = sys.argv[1]

if sys.argv[2] == 'high':
    print "##Loading high entropy data in memory..."
    MEMFILE = os.urandom(1000000000)
elif sys.argv[2] == 'low':
    print "##Loading low entropy data in memory..."
    MEMFILE = (b'\x00' * 1000000000)
else:
    print "##Wrong parameter, must be high / low..."
    exit(0)

with open( DEVICE , "wb" ) as dev:
    print "##Writing..."
    t = time.time()
    for i in xrange( 0, DATA_AMOUNT_GB ):
        dev.write( MEMFILE )
    total_time = time.time() - t


print "##Average throughput:", ( ( 1024 * DATA_AMOUNT_GB ) / total_time ), "MB/↘
    → s..."
```

A.9 Visualizzatore andamento scritture (wspeed.gplot)

Questo file viene usato da gnuplot, nel test di compressione, per tracciare un grafico avente sull'asse delle ascisse il tempo di scrittura (in minuti) e sull'asse delle ordinate il throughput (in MB/s).

```
#!/usr/bin/gnuplot
set terminal png size 1920,1080
set xlabel "time"
set ylabel "throughput"
set xdata time
set grid
#set timefmt "%H:%M:%S"
#set xrange ["00:00:00":"00:05:00"]
set yrange [0:]
set style line 2 lt 2 lw 5 lc rgb "blue"

set output "write_low_entropy.png"
set title "Write throughput, low entropy data"
set xrange [0:220]
plot "iostat-l.log" using 0:3 title "Write throughput" with lines ls 2

set output "write_high_entropy.png"
set title "Write throughput, high entropy data"
set xrange [0:220]
plot "iostat-h.log" using 0:3 title "Write throughput" with lines ls 2
```

A.10 Estrazione indirizzi zone azzerate (zero_stripes_finder.py)

Questo script confronta tra loro due immagini disco e stampa gli indirizzi di inizio e di fine di ogni zona che è stata azzerata.
Per eseguire lo script su due immagini IMG1.dd e IMG2.dd, lanciare il comando *python zero_stripes_finder.py IMG1.dd IMG2.dd*

```
import sys, operator, os,time
import numpy as np
from numpy import dtype

sector_size = 4096
f1 = open(sys.argv[1], "rb")
file1_size = os.path.getsize(sys.argv[1])
num_sector = file1_size/sector_size
instripe = False
stripe = []
percent = 0
try:
    for i in range(0,num_sector):
        if i%(num_sector/100) == 0:
            sys.stdout.write("\r%d%%" %percent)
            sys.stdout.flush()
            percent += 1
        #Read a sector and convert it to the correct format
        sector1 = f1.read(sector_size)
        conv_sec1 = np.fromstring(sector1, dtype=np.uint64)
        if np.any(conv_sec1 == 0):
            if instripe == False:
                begin = i
                instripe = True
        else:
            if instripe == True:
                instripe = False
                end = i
                if end - begin > 1000:
                    stripe.append([begin,end])
            else:
                instripe = False
finally:
    f1.close()
    for elem in stripe:
        print elem
```

Appendice B

Le implementazioni dei test

Questo appendice presenta gli script che abbiamo realizzato per eseguire i test proposti in questo lavoro in modo automatico. Ogni script genera un file di log il cui path può essere modificato cambiando il valore della costante LOGFILE all'inizio dei singoli script.

B.1 Test di stabilità dell'hash

Per eseguire questo script è necessario passare un unico parametro, ovvero il path del disco che si vuole testare (es.: sudo ./execute-hash.sh /dev/sdx). E' importante associare alla costante DISK_SIZE la capienza in GB del disco che si vuole testare, alla costante TEST_TIME il tempo in ore in cui mantenere il disco in idle e alla costante FILE_SIZE la dimensione in byte dei file con cui riempire il disco prima della formattazione.

```
#!/bin/bash

DEVICE=$1
DISK_SIZE=64 #in GB
TEST_TIME=16 #in hours
FILE_SIZE=100000000 #in bytes
LOGFILE=logfile.log;


function init_disk {

    echo -ne "Device $1 is going to be wiped..." | tee -a $LOGFILE; date >> ↘
        → $LOGFILE
    read -p "Are you sure? " -n 1 -r
    if [[ $REPLY =~ ^[Yy]$ ]]
    then
        sudo dc3dd wipe=$1
    else
        exit
    fi
```

```
    echo -ne "Formatting the drive..." | tee -a $LOGFILE; date >> $LOGFILE
    sudo mkfs.ext4 -F -E nodiscard $1 &> /dev/null

    echo -ne "Turning off write cache..." | tee -a $LOGFILE; date >> $LOGFILE
    sudo hdparm -W 0 $1 &> /dev/null
    echo -ne "Initialization completed..." | tee -a $LOGFILE; date >> $LOGFILE
}

#repeat the test at 25,50,75,100 %
BASE_SIZE=$((DISK_SIZE/4))
for ((i=$BASE_SIZE; i <= $DISK_SIZE ; i=i+$BASE_SIZE))
do
    init_disk $DEVICE
    echo -ne "Initialization complete..." | tee -a $LOGFILE; date >> $LOGFILE

    echo -ne "Filling the drive $DEVICE at $i%..." | tee -a $LOGFILE; date >> ↘
        → $LOGFILE
    sudo python filler.py $DEVICE $FILE_SIZE $i

    echo -ne "Starting comparer..." | tee -a $LOGFILE; date >> $LOGFILE
    sudo comparer.sh /dev/null $DEVICE &

    echo -ne "Formatting the drive..." | tee -a $LOGFILE; date >> $LOGFILE
    sudo mkfs.ext4 -F -E nodiscard $1 &> /dev/null

    echo -ne "Quick format completed!" | tee -a $LOGFILE; date >> $LOGFILE

    md5_presleep=$(md5sum $DEVICE | awk '{print $1}')
    sha1_presleep=$(sha1sum $DEVICE | awk '{print $1}')
    sleep $((TEST_TIME*3600))
    md5_poststleep=$(md5sum $DEVICE | awk '{print $1}')
    sha1_poststleep=$(sha1sum $DEVICE | awk '{print $1}')

if [ "$md5_presleep" = "$md5_postsleep" -a "$sha1_presleep" = "$sha1_postsleep" ↘
        → ]; then
    echo -ne "Hash remained stable during sleep!" | tee -a $LOGFILE; date >> ↘
        → $LOGFILE
else
    -ne "Hash did NOT remain stable during sleep. Warning: this disk cannot be ↘
        → used as evidence!" | tee -a $LOGFILE; date >> $LOGFILE
fi

    echo -ne "Killing comparer..." | tee -a $LOGFILE; date >> $LOGFILE
    sudo killall comparer.sh
done
```

B.2 Test dell'aggressività del garbage collector

Per eseguire questo script è necessario passare un unico parametro, ovvero il path del disco che si vuole testare (es.: sudo ./execute-hash.sh /dev/sdx). Le costanti sono le stesse utilizzate per il test sulla stabilità dell'hash: la costante DISK_SIZE rappresenta la capienza in GB del disco che si vuole testare, la costante TEST_TIME il tempo in ore da attendere per verificare l'attivazione o meno del garbage collector e la costante FILE_SIZE la dimensione in byte dei file con cui riempire il disco prima della formattazione.

```
#!/bin/bash

DEVICE=$1
DISK_SIZE=64 #in GB
TEST_TIME=16 #in hours
FILE_SIZE=100000000 #in bytes
LOGFILE=logfile.log;


function init_disk {

    echo -ne "Device $1 is going to be wiped..." | tee -a $LOGFILE; date >> ↘
        → $LOGFILE
    read -p "Are you sure? " -n 1 -r
    if [[ $REPLY =~ ^[Yy]$ ]]
    then
        sudo dc3dd wipe=$1
    else
        exit
    fi

    echo -ne "Formatting the drive..." | tee -a $LOGFILE; date >> $LOGFILE
    sudo mkfs.ext4 -F -E nodiscard $1 &> /dev/null

    echo -ne "Turning off write cache..." | tee -a $LOGFILE; date >> $LOGFILE
    sudo hdparm -W 0 $1 &> /dev/null
    echo -ne "Initialization completed..." | tee -a $LOGFILE; date >> $LOGFILE
}

#repeat the test at 25,50,75,100 %
BASE_SIZE=$((DISK_SIZE/4))
for ((i=$BASE_SIZE; i <= $DISK_SIZE ; i=i+$BASE_SIZE))
do
    init_disk $DEVICE
    echo -ne "Initialization complete..." | tee -a $LOGFILE; date >> $LOGFILE

    echo -ne "Filling the drive $DEVICE at $i%..." | tee -a $LOGFILE; date >> ↘
        → $LOGFILE
    sudo python filler.py $DEVICE $FILE_SIZE $i

    echo -ne "Starting comparer..." | tee -a $LOGFILE; date >> $LOGFILE
```

```
    sudo comparer.sh /dev/null $DEVICE &

    echo -ne "Formatting the drive..." | tee -a $LOGFILE; date >> $LOGFILE
    sudo mkfs.ext4 -F -E nodiscard $1 &> /dev/null

    echo -ne "Quick format completed!" | tee -a $LOGFILE; date >> $LOGFILE
    read -p "To do an idle test press I. To do a test with active overwrites ↘
        → press A. " -n 1 -r
    if [[ $REPLY =~ ^[Ii]$ ]]
    then
        sleep $((TEST_TIME*3600))
    elif [[ $REPLY =~ ^[Aa]$ ]]
    then
        START_TIME=$(date +%s)
        CURR_TIME=$(date +%s)
        TEST_TIME=$((TEST_TIME*3600))
        sudo mkdir -p /mnt/gctest
        sudo mount $1 /mnt/gctest
        for i in {1..128000} ; do echo -ne "EVIDENCE" >> knownfile.txt; done
        while [ $CURR_TIME -lt $[$START_TIME+$TEST_TIME] ]; do
            cp -v -f knownfile.txt /mnt/gctest/knownfile.txt
            CURR_TIME=$(date +%s)
        done
        sudo umount /mnt/gctest
    else
        echo -ne "Incorrect option..." | tee -a $LOGFILE; date >> $LOGFILE
        exit
    fi

    echo -ne "Killing comparer..." | tee -a $LOGFILE; date >> $LOGFILE
    sudo killall comparer.sh
done
```

B.3 Test della funzionalità di TRIM

Per eseguire questo script è necessario passare un unico parametro, ovvero il path del disco che si vuole testare (es.: sudo ./execute-trim.sh /dev/sdx). E' importante associare alla costante DISK_SIZE la capienza in GB del disco che si vuole testare e alla costante SECTORSIZE_DISK la dimensione dei settori del disco. E' inoltre possibile modificare altre costanti a inizio script per cambiare la dimensione dei file da scrivere (FILE_SIZE), la directory di mount (MOUNTDIR) e il tempo da attendere dopo la formattazione per aspettare che le funzionalità di TRIM o garbage collection azzerino i dati (SLEEP_AFTER_QUICK_FORMAT).

```
#!/bin/bash

DEVICE=$1
DISK_SIZE=64
FILE_SIZE=100000000
LOGFILE=logfile.log;
MOUNTDIR=mnt
SECTORSIZE_DISK=512
SLEEP_AFTER_QUICK_FORMAT=300

function init_disk {

    echo -ne "Device $1 is going to be wiped..." | tee -a $LOGFILE; date >> ↘
        → $LOGFILE
    read -p "Are you sure? " -n 1 -r
    if [[ $REPLY =~ ^[Yy]$ ]]
    then
        sudo dc3dd wipe=$1
    else
        exit
    fi

    echo -ne "Formatting the drive..." | tee -a $LOGFILE; date >> $LOGFILE
    sudo mkfs.ext4 -F -E nodiscard $1 &> /dev/null
    echo -ne "Turning off write cache..." | tee -a $LOGFILE; date >> $LOGFILE
    sudo hdparm -W 0 $1 &> /dev/null
    echo -ne "Initialization completed..." | tee -a $LOGFILE; date >> $LOGFILE
}

#repeat the test at 25,50,75,100 %
BASE_SIZE=$((DISK_SIZE/4))
for ((i=$BASE_SIZE; i <= $DISK_SIZE ; i=i+$BASE_SIZE))
do
    init_disk $DEVICE
    sudo mkdir $MOUNTDIR
    sudo mount $DEVICE $MOUNTDIR

    echo -ne "Filling the drive $DEVICE at $i%..." | tee -a $LOGFILE; date >> ↘
        → $LOGFILE
```

```
    sudo pyhton filler.py $MOUNTDIR $FILE_SIZE $i
    echo -ne "Starting comparer..." | tee -a $LOGFILE; date >> $LOGFILE
    sudo comparer.sh /dev/null $DEVICE &
    echo -ne "Formatting the drive..." | tee -a $LOGFILE; date >> $LOGFILE
    sudo mkfs.ext4 -F -E nodiscard $1 &> /dev/null

    ##Testing single file sanification (this cycle can be repeated to be sure)
    echo -ne "\n\nStarting single file sanification test..." | tee -a $LOGFILE; ↘
        → date >> $LOGFILE
    rand_file=$[ $RANDOM % ( ($i*1000000000) / $FILE_SIZE )
    start_sector=‘sudo hdparm --fibmap a | tail -1 | awk ’{print $2}’‘
    end_sector=‘sudo hdparm --fibmap a | tail -1 | awk ’{print $3}’‘
    rand_sector=$[ ($RANDOM % ($end_sector - $start_sector )) + $start_sector ↘
        → ]
    sudo rm -f $MOUNTDIR/$rand_file
    sync
    echo -ne "Waiting 10 seconds..."
    sleep 10
    #sudo hdparm --read-sector $rand_sector $DEVICE
    read_sector= ‘sudo dd if=$DEVICE bs=$SECTORSIZE_DISK count=1 skip= ↘
        → $rand_sector‘
    zeros=‘cmp -l -n $[ $SECTORSIZE_DISK - 1 ] /dev/zero $read_sector | wc -l | ↘
        → bc‘
    if [[ $zeros == 0 ]]
    then
        echo -ne "TRIM IS WORKING..." | tee -a $LOGFILE; date >> $LOGFILE
    else
        echo -ne "TRIM IS NOT WORKING..." | tee -a $LOGFILE; date >> $LOGFILE
    fi

    ## Testing quick format
    echo -ne "\n\nStarting quick format test..." | tee -a $LOGFILE; date >> ↘
        → $LOGFILE
    echo -ne "Formatting the drive..." | tee -a $LOGFILE; date >> $LOGFILE
    sudo mkfs.ext4 -F -E nodiscard $1 &> /dev/null

    sleep 10
    sudo umount $MOUNTDIR
    sleep $SLEEP_AFTER_QUICK_FORMAT
    echo -ne "Killing comparer..." | tee -a $LOGFILE; date >> $LOGFILE
    sudo killall -9 comparer.sh
    echo -ne "Test completed, take a look to the logfiles: $LOGFILE, comparer. ↘
        → log" | tee -a $LOGFILE; date >> $LOGFILE
done
```

B.4 Test di ricerca di pattern di azzeramento

Per eseguire questo script è necessario passare un unico parametro, ovvero il path del disco che si vuole testare (es.: sudo ./execute-pattern.sh /dev/sdx). E' possibile modificare le costanti a inizio script per cambiare il nome dell'immagine di output (IMG_DIFF), delle immagini disco generate prima (IMG_PRE) e dopo (IMG_POST) la formattazione, il tempo da attendere dopo la formattazione per aspettare che le funzionalità di TRIM o garbage collection azzerino i dati (TIMEOUT_AFTER_FORMAT), il numero di GB da scrivere (DATA_AMOUNT_GB) o la dimensione dei file da scrivere (FILE_SIZE).

```
#!/bin/bash

DEVICE=$1
LOGFILE=logfile.log
DATA_AMOUNT_GB=64
FILE_SIZE=1000000000
IMG_PRE=img_pre.dd
IMG_POST=img_post.dd
IMG_DIFF=diff.ppm
TIMEOUT_AFTER_FORMAT=60

function init_disk {

    echo -e "Device $1 is going to be wiped..." | tee -a $LOGFILE; date >> ↘
        → $LOGFILE
    read -p "Are you sure? " -n 1 -r
    if [[ $REPLY =~ ^[Yy]$ ]]
    then
        sudo dc3dd wipe=$1
    else
        exit
    fi

    #aggiungere formattazione ntfs
    echo -e "Formatting the drive..." | tee -a $LOGFILE; date >> $LOGFILE
    sudo mkfs.ext4 -F -E nodiscard $1 &> /dev/null

    echo -e "Turning off write cache..." | tee -a $LOGFILE; date >> $LOGFILE
    sudo hdparm -W 0 $1 &> /dev/null
    echo -e "Initialization completed..." | tee -a $LOGFILE; date >> $LOGFILE
}

init_disk $DEVICE
echo -e "Initialization complete..." | tee -a $LOGFILE; date >> $LOGFILE

echo -e "Filling the drive $DEVICE..." | tee -a $LOGFILE; date >> $LOGFILE
sudo pyhton filler.py $DEVICE $FILE_SIZE $DATA_AMOUNT_GB

echo -e "Acquiring disk image in $IMG_PRE..." | tee -a $LOGFILE; date >> ↘
        → $LOGFILE
sudo dc3dd if=$DEVICE of=$IMG_PRE
```

```
echo -e "Formatting the drive..." | tee -a $LOGFILE; date >> $LOGFILE
sudo mkfs.ext4 -F -E nodiscard $1 &> /dev/null

echo -e "Waiting for $TIMEOUT_AFTER_FORMAT s..." | tee -a $LOGFILE; date >> ↘
        → $LOGFILE
sleep $TIMEOUT_AFTER_FORMAT

echo -e "Acquiring disk image in $IMG_POST..." | tee -a $LOGFILE; date >> ↘
        → $LOGFILE
sudo dc3dd if=$DEVICE of=$IMG_POST

echo -e "Creating a graphical representation of the differences between the 2 ↘
        → images in $IMG_DIFF..." | tee -a $LOGFILE; date >> $LOGFILE
sudo python visualizer.py $IMG_PRE $IMG_POST $IMG_DIFF
```

B.5 Test di recuperabilità dei file cancellati

Questo script è stato strutturato a funzioni poichè il test con l'implementazione proprietaria di NTFS non può essere concluso completamente in Windows, a causa di alcune limitazioni intrinseche di Cygwin. Se si vuole testare tale file system è quindi necessario passare in seguito al quick format ad un sistema operativo basato su UNIX, limitandosi poi a chiamare la funzione post_format.
Lo script riceve come parametri il path del disco su cui eseguire il test, il path del file da utilizzare e il path dell'immagine disco che viene generata nel corso del test (es.: sudo ./execute-carving.sh /dev/sdx FILEPATH IMAGE_OUT). E' possibile cambiare la quantità di GB da scrivere modificando la costante DATA_AMOUNT_GB a inizio script.

```
#!/bin/bash

DEVICE=$1
FILE=$2
IMAGE_OUT=$3
LOGFILE=logfile.log
DATA_AMOUNT_GB=30
FILE_SIZE=$(stat $2 | nawk '/Size/ {print $2}')

function init_disk {

    echo -e "Device $1 is going to be wiped..." | tee -a $LOGFILE; date >> ↘
        → $LOGFILE
    read -p "Are you sure? " -n 1 -r
    if [[ $REPLY =~ ^[Yy]$ ]]
    then
        sudo dc3dd wipe=$1
    else
        exit
    fi

    echo -e "Formatting the drive..." | tee -a $LOGFILE; date >> $LOGFILE
    sudo mkfs.ext4 -F -E nodiscard $1 &> /dev/null

    echo -e "Turning off write cache..." | tee -a $LOGFILE; date >> $LOGFILE
    sudo hdparm -W 0 $1 &> /dev/null
    echo -e "Initialization completed..." | tee -a $LOGFILE; date >> $LOGFILE
}

function pre_format {
    init_disk $DEVICE
    echo -e "Calculating hash..." | tee -a $LOGFILE; date >> $LOGFILE
    rm hashlist.txt
    md5sum $FILE | awk '{print $1}' | tee -a hashlist.txt
    sha1sum $FILE | awk '{print $1}' | tee -a hashlist.txt
    echo -e "Filling the drive..." | tee -a $LOGFILE; date >> $LOGFILE
    for ((i=1; i <= $(($DATA_AMOUNT_GB/$FILE_SIZE)); i++))
```

```
    do
        cp -v -f $FILE "$i.jpg"
    done
    echo -e "Format the disk with a windows OS to test NTFS or continue to test ↘
        → EXT4..." | tee -a $LOGFILE; date >> $LOGFILE
    echo -e "Do you want to continue?" | tee -a $LOGFILE; date >> $LOGFILE
    if [[ $REPLY =~ ^[Yy]$ ]]
    then
        echo -e "Formatting the drive..." | tee -a $LOGFILE; date >> $LOGFILE
        sudo mkfs.ext4 -F -E nodiscard $1 &> /dev/null
    else
        exit
    fi
}

function post_format {
    echo -e "Extracting disk image, saved in $IMAGE_OUT..." | tee -a $LOGFILE; ↘
        → date >> $LOGFILE
    sudo dc3dd if=$DEVICE of=$IMAGE_OUT
    echo -e "Carving image..." | tee -a $LOGFILE; date >> $LOGFILE
    carver.sh $IMAGE_OUT
    echo -e "Analysis completed" | tee -a $LOGFILE; date >> $LOGFILE

}

pre_format
post_format
```

B.6 Test di verifica della presenza di compressione

Per eseguire questo script è necessario passare un unico parametro, ovvero il path del disco che si vuole testare (es.: sudo ./execute-compression.sh /dev/sdx).

```
#!/bin/bash

DEVICE=$1
LOGFILE=logfile.log;
GNUPLOT_SCRIPT=wspeed.gplot

function init_disk {
    echo -e "Turning off write cache...\n" | tee -a $LOGFILE; date >> $LOGFILE
    sudo hdparm -W 0 $1 &> /dev/null

    echo -e "Device $1 is going to be wiped..." | tee -a $LOGFILE; date >> ↘
        → $LOGFILE
    read -p "Are you sure? " -n 1 -r
    if [[ $REPLY =~ ^[Yy]$ ]]
    then
        sudo dc3dd wipe=$1
    else
        exit
    fi

    #echo -e "\nFormatting the drive..." | tee -a $LOGFILE; date >> $LOGFILE
    #sudo mkfs.ext4 -F -E nodiscard $1 &> /dev/null

    echo -e "Initialization completed..." | tee -a $LOGFILE; date >> $LOGFILE
}

init_disk $DEVICE
echo -e "Starting to sample the throughput with iostat..." | tee -a $LOGFILE; ↘
        → date >> $LOGFILE
iostat -dmxt $DEVICE 1 | nawk '/sd/ {i=i+1; print i,$6,$7 >>"iostat-l.log"}' &
echo -e "Writing low entropy file..." | tee -a $LOGFILE; date >> $LOGFILE
sudo python write-benchmark.py $DEVICE low
sudo killall iostat

init_disk $DEVICE
echo -e "Starting to sample the throughput with iostat..." | tee -a $LOGFILE; ↘
        → date >> $LOGFILE
iostat -dmxt $DEVICE 1 | nawk '/sd/ {i=i+1; print i,$6,$7 >>"iostat-h.log"}' &
echo -e "\nWriting high entropy file..." | tee -a $LOGFILE; date >> $LOGFILE
sudo python write-benchmark.py $DEVICE high
sudo killall iostat
echo -e "\nPlotting data with gplot..." | tee -a $LOGFILE; date >> $LOGFILE
sudo gnuplot $GNUPLOT_SCRIPT
```

B.7 Test di verifica degli attributi S.M.A.R.T.

Per eseguire questo script è necessario passare un unico parametro, ovvero il path del disco che si vuole testare (es.: sudo ./execute-compression.sh /dev/sdx). Per quanto riguarda le costanti a inizio script, è possibile modificare la percentuale di workload cambiando il valore della costante WORKLOAD e il tempo di esecuzione del test (in minuti) cambiando il valore della costante TEST_TIME.

```
#!/bin/bash

DEVICE=$1;
LOGFILE=logfile.log;
WORKLOAD=50;  #in percent
TEST_TIME=120; #in minutes

#disable write cache
echo -e "\nTurning off write cache..." | tee -a $LOGFILE; date >> $LOGFILE
sudo hdparm -W 0 $DEVICE &> /dev/null

echo -e "Saving current S.M.A.R.T. values..." | tee -a $LOGFILE; date >> ↘
        → $LOGFILE
logical_pre=‘sudo smartctl -A $DEVICE | grep 233 |  awk ’{print $10}’‘
physical_pre=‘sudo smartctl -A $DEVICE | grep 241 |  awk ’{print $10}’‘
echo -e "##$logical_pre, ##$physical_pre" | tee -a $LOGFILE; date >> $LOGFILE

echo "Generating workload for the next $TEST_TIME minutes..." | tee -a $LOGFILE ↘
        → ; date >> $LOGFILE

#generate workload for DEVICE for TIME in minutes, INTENSITY percent
sudo python workload_generator.py $DEVICE $WORKLOAD $((TEST_TIME*60))

echo "Workload simulation finished..." | tee -a $LOGFILE; date >> $LOGFILE
echo "Calculating new S.M.A.R.T. values..." | tee -a $LOGFILE; date >> $LOGFILE

logical_post=‘sudo smartctl -A $DEVICE | grep 233 |  awk ’{print $10}’‘
physical_post=‘sudo smartctl -A $DEVICE | grep 241 |  awk ’{print $10}’‘
diff_logical=$((logical_post-logical_pre))
diff_physical=$((physical_post-physical_pre))

echo -e "Logical writes: $diff_logical" | tee -a $LOGFILE; date >> $LOGFILE
echo -e "Physical writes: $diff_physical" | tee -a $LOGFILE; date >> $LOGFILE
```

B.8 Test black box delle funzionalità di wear leveling

L'implementazione del test black box delle funzionalità di wear leveling segue tutti i passi del protocollo fino all'acquisizione dell'immagine disco, ma non si occupa di automatizzare carving e analisi dei dati recuperati. E' stato scelto questo approccio poichè, nel caso in cui sia possibile recuperare dati duplicati, è altamente probabile che il carver li estragga in forma frammentata. Un controllo automatico basato su hash non sarebbe quindi significativo, poiché i frammenti potrebbero contenere dati comprensibili anche nel caso in cui l'hash risulti differente da quello del file originale. Per chi volesse eseguire questo test è quindi consigliabile controllare manualmente i dati recuperati.
Per eseguire questo script è necessario passare un unico parametro, ovvero il path del disco che si vuole testare (es.: sudo ./execute-compression.sh /dev/sdx). E' importante associare alla costante DISK_SIZE la capienza in GB del disco che si vuole testare, e se si desidera modificare il numero di sovrascritture è sufficiente cambiare il valore della costante OVERWRITES.

```
#!/bin/bash

DEVICE=$1
DISK_SIZE=64 #in GB
OVERWRITES=10000
LOGFILE=logfile.log;

function init_disk {

    echo -ne "Device $1 is going to be wiped..." | tee -a $LOGFILE; date >> ↘
        → $LOGFILE
    read -p "Are you sure? " -n 1 -r
    if [[ $REPLY =~ ^[Yy]$ ]]
    then
        sudo dc3dd wipe=$1
    else
        exit
    fi

    echo -ne "Formatting the drive..." | tee -a $LOGFILE; date >> $LOGFILE
    sudo mkfs.ext4 -F -E nodiscard $1 &> /dev/null

    echo -ne "Turning off write cache..." | tee -a $LOGFILE; date >> $LOGFILE
    sudo hdparm -W 0 $1 &> /dev/null
    echo -ne "Initialization completed..." | tee -a $LOGFILE; date >> $LOGFILE
}

#repeat the test at 25,50,75 %
BASE_SIZE=$((DISK_SIZE/4))
for ((i=$BASE_SIZE; i<=$((3*$DISK_SIZE/4)); i=i+$BASE_SIZE))
do
    init_disk $DEVICE
```

```
    echo -ne "Initialization complete..." | tee -a $LOGFILE; date >> $LOGFILE

    echo -ne "Filling the drive $DEVICE at $i%..." | tee -a $LOGFILE; date >> ↘
        → $LOGFILE
    sudo python filler.py $DEVICE 100000000 $i

    echo -ne "Starting comparer..." | tee -a $LOGFILE; date >> $LOGFILE
    sudo comparer.sh /dev/null $DEVICE &

    sudo mkdir -p /mnt/wltest
    sudo mount $1 /mnt/wltest
    for i in {1..128000} ; do echo -ne "EVIDENCE" >> knownfile.txt; done
    COUNT=0
    cp -v -f knownfile.txt /mnt/wltest/knownfile.txt
    PREV_START_LBA=$(sudo hdparm --fibmap /mnt/wltest/knownfile.txt | awk 'FNR ↘
        → == 4 {print $2}')
    PREV_END_LBA=$(sudo hdparm --fibmap /mnt/wltest/knownfile.txt | awk 'FNR == ↘
        →  4 {print $3}')
    while [ "$COUNT" -lt "$OVERWRITES" ]; do
        cp -v -f knownfile.txt /mnt/wltest/knownfile.txt
        let "COUNT += 1"
        CURR_START_LBA=$(sudo hdparm --fibmap /mnt/wltest/knownfile.txt | awk ' ↘
        → FNR == 4 {print $2}')
        CURR_END_LBA=$(sudo hdparm --fibmap /mnt/wltest/knownfile.txt | awk ' ↘
        → FNR == 4 {print $3}')
        if [ "$PREV_START_LBA" != "$CURR_START_LBA" -o "$PREV_END_LBA" != " ↘
        → $CURR_END_LBA" ]; then
            echo -ne "File position changed!" | tee -a $LOGFILE; date >> ↘
        → $LOGFILE
            echo -ne "Previous start LBA: $PREV_START_LBA, current start LBA: ↘
        → $CURR_START_LBA." | tee -a $LOGFILE; date >> $LOGFILE
            echo -ne "Previous end LBA: $PREV_END_LBA, current end LBA: ↘
        → $CURR_END_LBA." | tee -a $LOGFILE; date >> $LOGFILE
            PREV_START_LBA=$CURR_START_LBA
            PREV_END_LBA=$CURR_END_LBA
        fi
    done
    sudo umount /mnt/wltest

    echo -ne "Killing comparer..." | tee -a $LOGFILE; date >> $LOGFILE
    sudo killall comparer.sh

    echo -e "Extracting disk image, saved in img-$i-GB.dd..." | tee -a $LOGFILE ↘
        → ; date >> $LOGFILE
    sudo dc3dd if=$DEVICE of=img-"$i"-GB.dd
done
```

Bibliografia

[1] Neal Ekker et al. An introduction to Solid State Storage. SNIA https://members.snia.org/apps/group_public/download.php/35796/SSSI%20Wht%20Paper%20Final.pdf, 2009.

[2] Jitu J. Makwana and Dr. Dieter K. Schroder. A non volatile memory overview. http://aplawrence.com/Makwana/nonvolmem.html, 2004.

[3] Roberto Bez et al. Introduction to flash memory. *Proceedings of the IEEE, Vol. 91 No. 4*, page 489–502, 2003.

[4] Esther Spanjer. Flash management - why and how? A detailed overview of flash management techniques. SMART Modular Technologies Corporation, 2009.

[5] Christopher King and Timothy Vidas. Empirical analysis of solid state disk data retention when used with contemporary operating systems. 11th Digital Forensic Research Workshop annual conference, 2011.

[6] Marina Mariano. ECC options for improving NAND flash memory reliability. http://www.micron.com/~/media/Documents/Products/Software%20Article/SWNL_implementing_ecc.pdf, 2012.

[7] TN-29-63. TN-29-63: Error correction code (ECC) in SLC NAND. Micron Technology Inc. http://www.micron.com/~/media/Documents/Products/Technical%20Note/NAND%20Flash/tn2963_ecc_in_slc_nand.pdf, 2011.

[8] TN-29-59. TN-29-59: Bad block management in NAND flash memory. Micron Technology Inc. Technical Note. http://www.micron.com/~/media/Documents/Products/Technical%20Note/NAND%20Flash/tn2959_bbm_in_nand_flash.pdf, 2011.

[9] AP-684. AP-684: Understanding the flash translation layer (FTL) specification. Intel Application Note. http://staff.ustc.edu.cn/~jpq/paper/flash/2006-Intel%20TR-Understanding%20the%20flash%20translation%20layer%20%28FTL%29%20specification.pdf, 1998.

[10] Graeme B. Bell and Richard Boddington. Solid state drives: The beginning of the end for current practice in digital forensic recovery? *The Journal of Digital Forensics Security and Law, Vol. 5 No. 3*, 2010.

[11] Christopher J. Antonellis. Solid state disks and computer forensics. *ISSA Journal July 2008*, page 36–38, 2008.

[12] Micheal Wei et al. Reliably erasing data from flash-based solid state drives. University of California, San Diego. `http://cseweb.ucsd.edu/users/m3wei/assets/pdf/FMS-2010-Secure-Erase.pdf`, 2010.

[13] Trevor Bunker et al. Ming II: A flexible platform for NAND flash-based research. UCSD CSE Technical Report CS2012-0978, 2012.

[14] Marcel Breeuwsma et al. Forensic data recovery from flash memory. *Small Scale Digital Device Forensics Journal, Vol. 1 No. 1*, 2007.

[15] David Billard and Rolf Hauri. Making sense of unstructured flash-memory dumps. *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 1579–1583, 2010.

[16] B. J. Phillips et al. Recovering data from USB flash memory sticks that have been damaged or electronically erased. *Proceedings of the 1st International Conference on Forensic Applications and Techniques in Telecommunications, Information, and Multimedia*, 2008.

[17] Robert Templeman and Apu Kapadia. GANGRENE: Exploring the mortality of flash memory. 7th USENIX Workshop on Hot Topics in Security (HotSec), 2012.

[18] James Luck and Mark Stokes. An integrated approach to recovering deleted files from NAND flash data, 2008.

[19] Sergei Skorobogatov. Data remanence in flash memory devices. *Proceedings of the 7th International Workshop on Cryptographic Hardware and Embedded Systems*, 2005.

[20] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal, Vol. 27 No. 3*, page 379–423, 1948. `http://www.alcatel-lucent.com/bstj/vol27-1948/articles/bstj27-3-379.pdf`.

[21] TN-29-42. TN-29-42: Wear leveling techniques in NAND flash devices. Micron Technology Inc. Technical Note. `http://www.micron.com/~/media/Documents/Products/Technical%20Note/NAND%20Flash/tn2942_nand_wear_leveling.pdf`, 2008.

[22] TN-29-61. TN-29-61: Wear leveling in micron® NAND flash memory. Micron Technology Inc. Technical Note. `http://www.micron.com/~/media/Documents/Products/Technical%20Note/NAND%20Flash/tn2961_wear_leveling_in_nand.pdf`, 2011.

[23] Jen-Wei Hsieh Yuan-Hao Chang and Tei-Wei Kuo. Improving flash wear leveling by proactively moving static data. *IEEE Transactions on Computers, Vol. 59 No. 1*, pages 53–65, 2010.

[24] Xiao-Yu Hu et al. Write amplification analysis in flash-based solid state drives. *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, 2009. `http://doi.acm.org/10.1145/1534530.1534544`.