Forensic Acquisition

Digital Forensics - ay 2024/2025

Enrico RUSSO <enrico.russo@dibris.unige.it>

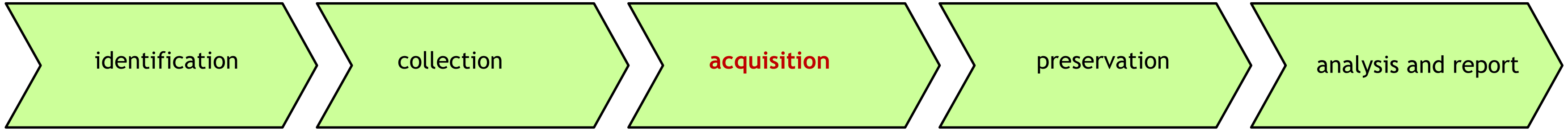
Outline

- Introduction to forensic acquisition
- HDD and SSD technologies
- Common interface standards and protocols
- Disk encryption: Bitlocker
- Physical and logical acquisitions
- Toolset and examples
    - dd, dc3dd, Expert Witness Format, Advanced Forensic Format, Guymager, FTK Imager

Forensic Acquisition

Acquisition is the process of cloning or copying digital data evidence.

- **forensically sound** (**integrity** and **non-repudiation**)
  - the copies must be **identical** to the **original**
  - the procedures must be **documented** and **implemented using known methods and technologies**, so that they can be verified by the opposite party.
- a **critical** step
  - **proper handling of data** ensures that all actions taken on it can be **checked**, **repeated**, and **verified** at **any time**
  - **incomplete or incorrect handling of data** has the potential to **compromise** the entire investigation



digital forensics process

Forensic Acquisition: critical concepts

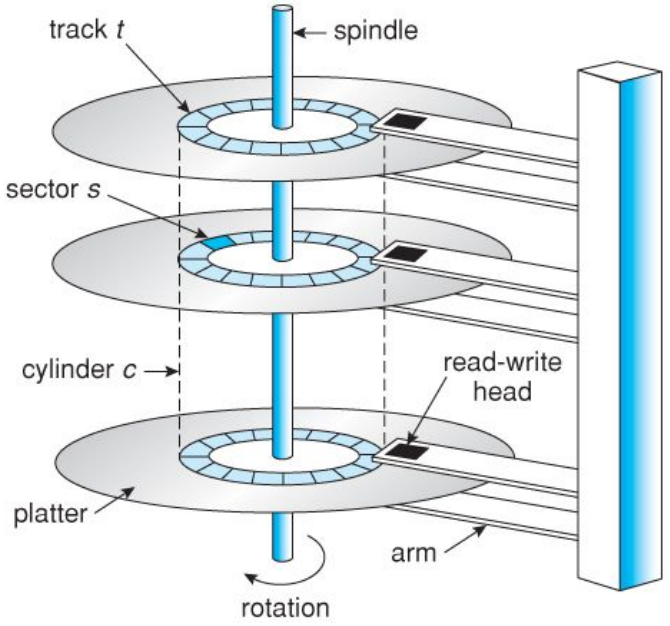
- It is generally recommended to **avoid conducting analysis** on the **original device** (namely, **best evidence**)
- Creating a **forensic image** is typically considered the most effective method for preserving digital evidence
    - One or more, **usually two**
- Accessing the original media only once during the acquisition phase can help **minimize the risk** of altering or damaging the evidence

This lesson focuses on **hard disks**, which are frequently encountered as a **primary source** of **digital evidence**.

Hard Disk technology: magnetic disks

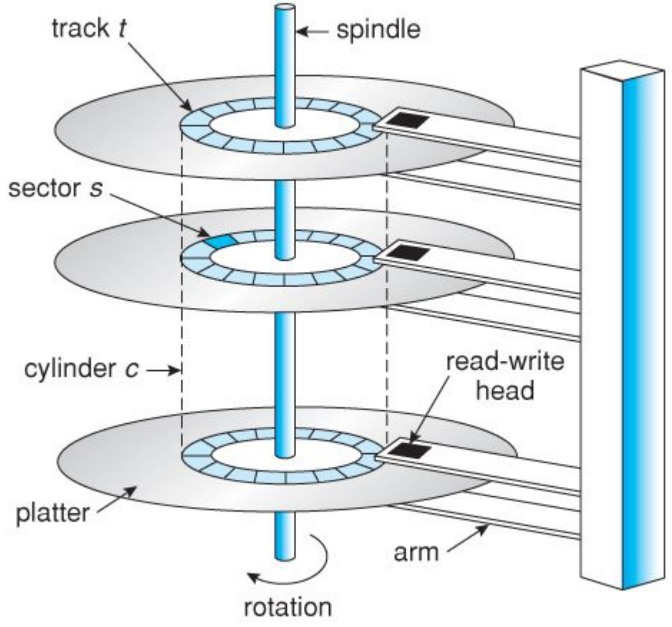
A hard disk is a sealed unit containing a number of **platters** in a stack

- each platter has two working **surfaces**
- each working surface is divided into a number of concentric rings called **tracks**
    - the collection of all tracks that are the same distance from the edge of the platter is called a **cylinder**.



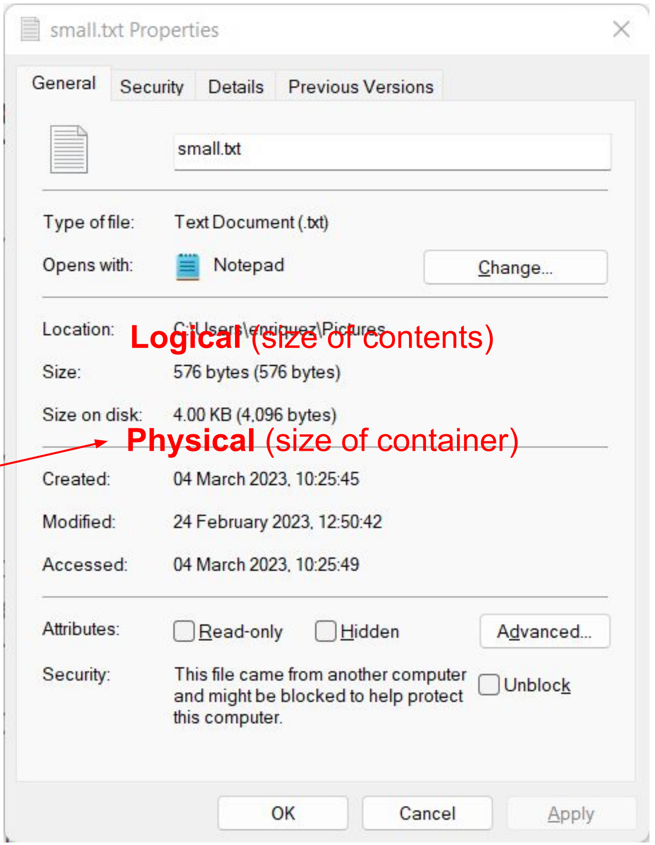
Magnetic disks [2]

- each track is further divided into **sectors**
  - a sector is the **smallest unit** that can be accessed on a **storage device**
  - traditionally containing **512 bytes** of data each, but recent hard drives have switched to **4KB** sectors (**Advanced Format**)
  - a **cluster** is a group of sectors (from 1 to 64 sectors) that make up the **smallest unit of disk** allocation for a file within a **file system**
- the data on a hard drive is read by read-write **heads**
  - the standard configuration uses one head per surface and they moves simultaneously from one cylinder to another



Magnetic disks [2]

- each track is further divided into **sectors**
  - a sector is the **smallest unit** that can be accessed on a **storage device**
  - traditionally containing **512 bytes** of data each, but recent hard drives have switched to **4KB** sectors (**Advanced Format**)
  - a **cluster** is a group of sectors (from 1 to 64 sectors) that make up the **smallest unit of disk** allocation for a file within a **file system**
- the data on a hard drive is read by read-write **heads**
  - the standard configuration uses one head per surface and they moves simultaneously from one cylinder to another


small.txt Properties

small.txt

Type of file: Text Document (.txt)
Opens with: Notepad

Location: C:\Users\enriquez\Pictures
Size: 576 bytes (576 bytes)
Size on disk: 4.00 KB (4,096 bytes)

Logical (size of contents)
Physical (size of container)

Created: 04 March 2023, 10:25:45
Modified: 24 February 2023, 12:50:42
Accessed: 04 March 2023, 10:25:49

Attributes: ☐ Read-only ☐ Hidden

Security: This file came from another computer and might be blocked to help protect this computer. ☐ Unblock

Magnetic disks: low level format

- A low level format is performed on the blank platters to create data structures for tracks and sectors
  - creates all of the headers and trailers marking the beginning and ends of each sector
  - header and trailer also keep the **linear sector numbers** (cf. LBA later), and **error-correcting codes** (**ECC**)
- All disks are shipped with a few bad sectors (additional ones can be expected to go bad slowly over time)
  - disks keep spare sectors to replace bad ones
  - ECC calculation is performed with every disk read or write:
    - if an error is detected but the data is recoverable, then a **soft error** has occurred
    - if the data on a bad sector cannot be recovered, then a **hard error** has occurred. A bad sector can be replace with a spare one (but any information written is usually lost)

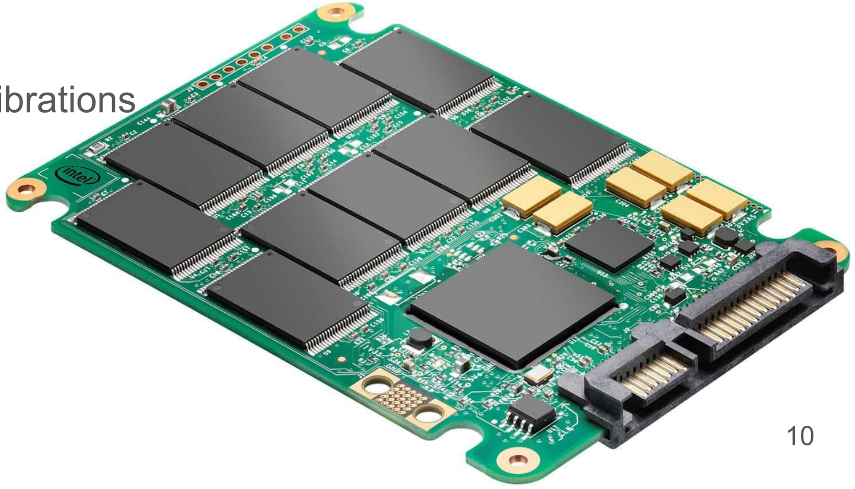
Magnetic disks: addressing the sectors

- Number of cylinders (tracks), heads (sides), and sectors (aka CHS) uniquely identify the **physical geometry** of a disk
- Each sector is given an address, starting at 1 for each track
  - we can address a specific sector by using the cylinder address (C) to get the track, the head number (H) to get the platter and side, and the sector address (S) to get the sector in the track ⇒ **CHS addressing**
- CHS addressing has been proven to be too limiting and is **not used anymore** (CHS works for drives up to 504 MB in capacity, ECHS up to 8 GB)
- **Logical Block Address** (LBA) is the standard: use a single number, starting at 0, to address each sector

Solid State Drives (SSD)

Like an HDD, an SSD is a **nonvolatile storage** device that stores data whether or not it is connected to power.

- An HDD, however, uses magnetic spinning platters
- SSDs use **flash memory chips**
    - faster
    - non-moving parts and resistant to shocks and vibrations
    - shorter lifespan and more expensive



SSD layout

- the smallest unit of an SSD is a **page**, which is composed of several memory cells
    - the page sizes are 2KB, 4KB, 8KB, 16KB or larger (usually they are 4 KB in size).
- Several pages on the SSD are summarized to a **block**
    - the block size typically varies between 256KB (128 pages * 2KB per page) and 4MB (256 pages * 16KB per page)

SSD: Reading and Writing Data

Reading and writing data on SSDs can be performed under the above constraints

Operation	**Description**
Read and Write	An SSD can read and write data at the **page level**
Write	Writing is only possible if **other pages in the block are empty**. Otherwise, it leads to **write amplification**
Erase Data	An SSD can **only erase an entire block** at once due to the physical and electrical characteristics of the memory cells

SSD: Program/Erase Cycle

Modifying data requires a **Program/Erase (P/E) cycle**.

- During a P/E cycle, an **entire block** containing the targeted pages is **written to memory**
- The block is then **marked for deletion**, and the updated data is **rewritten to another block**

SSD: Optimization and SSD Endurance

- Garbage Collection & Performance Optimization
    - The **erase** operation does not happen **immediately after data is marked for deletion**. Instead, the SSD performs it **asynchronously** when necessary to optimize performance
    - **Garbage Collection** helps free up space efficiently while minimizing interruptions to read/write operations
- Wear Leveling & SSD Lifespan
    - Every flash **memory cell** has a **limited number of P/E cycles** before it wears out
    - **Wear leveling** is a technique used by SSDs to **distribute writes** evenly across all memory blocks

SSD: TRIM

- many file systems handle delete operations by flagging data blocks as "not in use" that can be overwritten when required
  - magnetic disks: an overwrite of existing data is no different from writing into an empty sector
  - SSD: this behavior may worsen the overhead of write amplification (e.g., unnecessary copying of invalid data pages) and increase P/E cycles
- an **operating system** can uses the **TRIM** command to notify the SSD which data pages no longer contain valid data, allowing the SSD to manage storage space efficiently

SSD: Digital Forensics

- **TRIM wipes data** by forcing the garbage collector to run and makes **recovering digital evidence impossible**
- garbage collector execution is **managed internally** by the controller of the SSD without any possibility of interference from the host
  - a hardware write blocker (see later) can not avoid it
  - the **data may change** midway or between two **different acquisitions**

Accessing the disk content

Disk content is available through (different) standard interfaces. Each standard defines how the disk controller can

- **physically** connect the disk with a specific connector and cable
- **logically** connect (and interact with) the disk by using a shared command and transport protocol that defines how data is transferred (e.g., ATA - Advanced Technology Attachment)

In general, each new standard adds a faster method of reading and writing data or fixes a limitation (e.g., size of disks) of a previous standard.

Common interface standards

- **ATA** (Advanced Technology Attachment):
  - different standards ATA-1, ATA-2, …, ATA-8
  - includes specification for removable media, namely ATAPI (but it carries SCSI commands and responses through the ATA, see below)
  - Parallel ATA (P-ATA): old and a.k.a. Integrated Device Electronics (IDE)
  - Serial ATA (SATA): used nowadays and introduced from ATA/ATAPI-7. ATA-8 introduces features for SSD (e.g., TRIM)
    - SATA revision 3.0 (6 Gbit/s, Serial ATA-600)
- **SCSI** (Small Computer System Interface)
  - now replaced by SAS (Serial Attached SCSI) that is based on the SCSI standard, but uses a serial interface to connect storage
    - better scalable and faster (supports data transfer rates of up to 24 Gbit/s)

Common interface standards [2]

- **NVMe** (Non-Volatile Memory Express):
  - uses a PCIe interface to connect storage devices to a computer.
  - commonly used for high-performance solid-state drives since it supports data transfer rates of up to 32 GB/s
- **USB** (Universal Serial Bus):
  - uses a serial interface to connect storage devices to a computer
  - **mass storage** is the standard protocol used for storage devices
  - commonly used for external hard drives, flash drives, and other portable storage devices
  - USB 3.1 Gen 1 standard supports speeds up to 5 Gbit/s, while the USB 3.1 Gen 2 standard, supports speeds up to 20 Gbit/s.

ATA specification: Hard Disk Passwords

ATA-3 introduced HD password as an optional security feature.

- two passwords: **user** and **master**
- the **master password** was designed so an **administrator** can gain access in case the user password was lost (every hard disk is initially supplied with an *undocumented* master password)
- if passwords are being used, there are two modes that the disk can operate:
  - **high security mode**: the user and master password can unlock the disk
  - **maximum-security mode**: the user password can unlock the disk but the master password can unlock the disk after the disk content have been wiped
- It’s a lock mechanism, <u>no encryption</u>

ATA specification: Hard Disk Passwords [2]

- a protected HD will require the **SECURITY_UNLOCK** command to be executed with the correct password before any other ATA command
    - after the password has been entered, the disk works normally until the disk is powered on
- some ATA commands are still enabled on the HD when it is locked (so it may show as a valid disk when it is connected to a computer)
    - however, trying to read data from a locked disk will produce an error
- Passwords can be set through the BIOS or specific software applications (e.g., Linux users have `hdparm`)

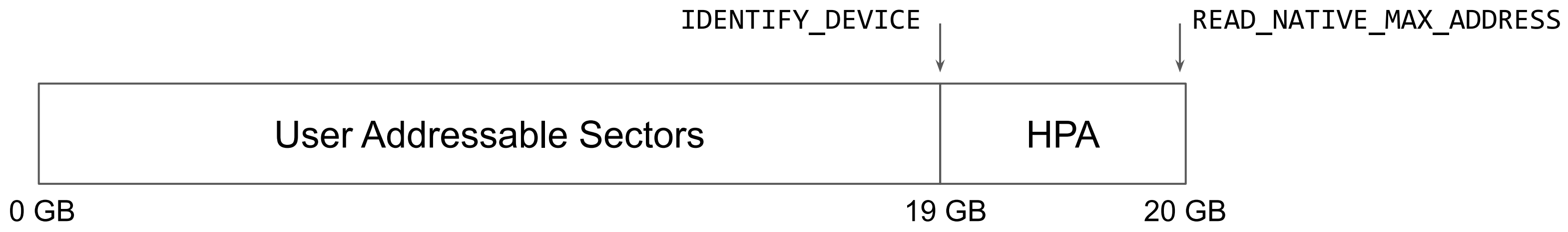
ATA specification: Host Protected Area

The **Host Protected Area** (HPA) was added in ATA-4 with the goal of providing a location not visible to a computer's operating system (OS).

- for example, a vendor can store the necessary files to install or recover the OS, i.e., perform a factory reset
- the HPA is at the **end of the disk**
  - when used, it can be accessed by reconfiguring the hard disk

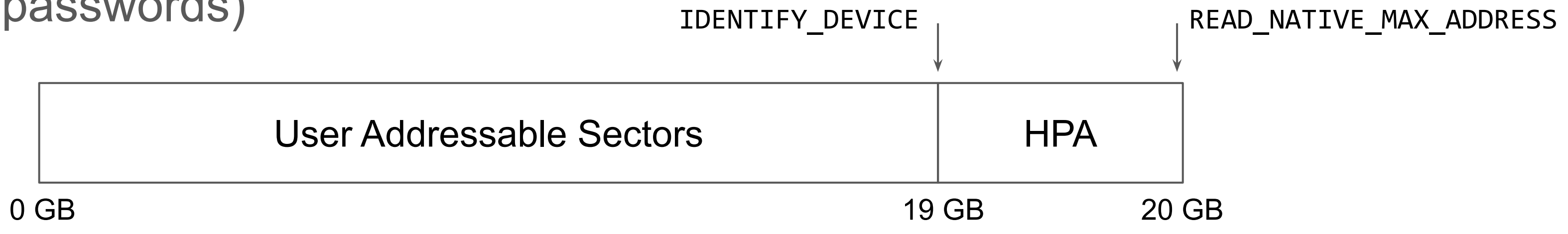
ATA specification: Host Protected Area

- Two ATA commands that return maximum physical addressable sectors
  - **`READ_NATIVE_MAX_ADDRESS`**: return the maximum physical address
  - **`IDENTIFY_DEVICE`**: return only the number of sectors that a user can access
- To create an HPA, the `SET_MAX_ADRESS` command is used to set the maximum address to which the user should have access (to remove it, use `SET_MAX_ADDRESS` = `READ_NATIVE_MAX_ADDRESS`)



ATA specification: Host Protected Area [2]

- the `SET_MAX_ADDRESS` command support different settings, e.g.,
  - **volatility bit**: the HPA exist after the hard disk is reset or power cycled (otherwise the effect is permanent)
  - **locking command**: prevents modification to the maximum address until next reset
- when the BIOS requires to read/write some data in the HPA
  - it uses SET_MAX_ADDRESS with volatility bit and locking.
- It is possible to protect settings with a password (different from HD passwords)



ATA specification: Device Configuration Overlay

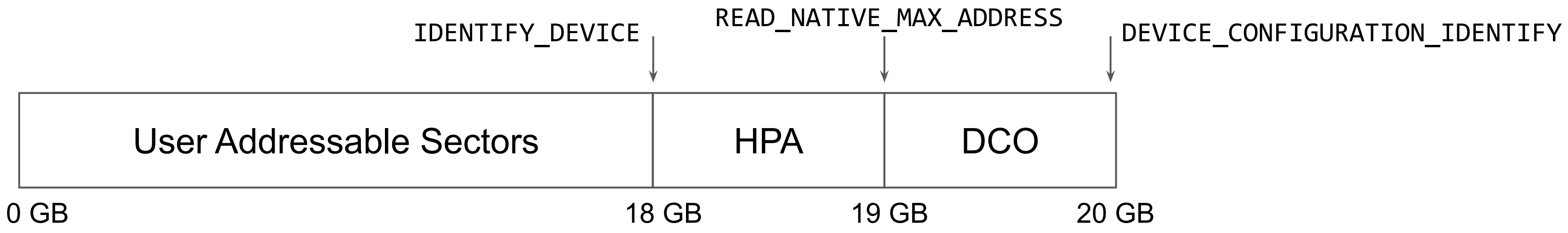
The Device Configuration Overlay (DCO) feature was first introduced in ATA-6 standard and allows the **apparent capabilities** of an HD to be **limited**.

- DCO was introduced to allow older systems to access newer HD (allowing manufacturers to build one version of the product to satisfy all the needs)
  - for example, old ATA standards used 28-bit addressing to specify the location of data on the hard drive, which allowed for a maximum capacity of 137 gigabytes
- A computer uses the IDENTIFY_DEVICE command to determine which features an HD support
  - a DCO can cause the above command to show that supported features are not support and show a smaller disk size (using `DEVICE_CONFIGURATION_SET`/`DEVICE_CONFIGURATION_RESET`)
- DCO can be also used to hide data

Co-existence of HPA and DCO

The DCO and HPA can co-exist on the same HDD (but DCO must be set first)

- the `DEVICE_CONFIGURATION_IDENTIFY` command return the actual features and size of a disk
  - we can detect DCO if `DEVICE_CONFIGURATION_IDENTIFY` ≠ `IDENTIFY_DEVICE`
  - we can detect an HPA that hides sectors if `READ_NATIVE_MAX_ADDRESS` ≠ `DEVICE_CONFIGURATION_IDENTIFY`



HPA and DCO: Investigative Significance

Forensics investigators need to be aware of these two areas

- an end user can modify and write to the HPA and DCO, allowing them to potentially **hide data**
- the HPA and DCO are **hidden** from the OS, BIOS, and the user
  - forensic tools typically **support** or **partially support** the **detection** of HPA and DCO (partial support involves the capability of detecting when the HPA and DCO are active simultaneously)
  - at least three different methods for detecting HPA on Linux: `dmesg`, `hdparm`, and `disk_stat` (https://wiki.sleuthkit.org/index.php?title=Disk_stat)

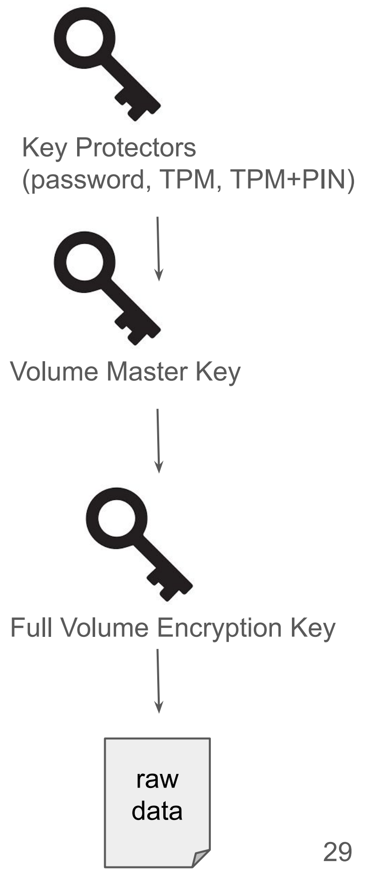
Disk Encryption: Bitlocker

BitLocker (Windows) is one of the most advanced and most commonly used solutions for **full disk encryption** (i.e., it protects information by encrypting all of the data on a disk).

- it makes use of **symmetric encryption** (by default, AES-128)
- on modern systems, it is coupled with a **Trusted Platform Module** (TPM)
    - the main functions of TPM are the generation, storage and secure management of cryptographic keys
    - on a computer without TPM a password can be used (then BitLocker encryption will be just as secure as the password you set)

Bitlocker: under the hood

- it uses different **symmetric keys**:
  - raw data is encrypted with the **Full Volume Encryption Key** (FVEK)
  - FVEK is then encrypted with the **Volume Master Key** (VMK)
  - VMK is in turn encrypted by one of several possible methods depending on the chosen authentication type (that is, **key protectors** or TPM) and recovery scenarios
- the use of **intermediate key** (VMK between FVEK and any key protectors) allows **changing the keys** without **the need to re-encrypt** the raw data in a case a given key protector is compromised or changed
  - When changing a key protector, a new VMK will be created and used to encrypt the old FVEK with the new VMK
- encrypted FVEK and VMK are **stored in the encrypted drive**

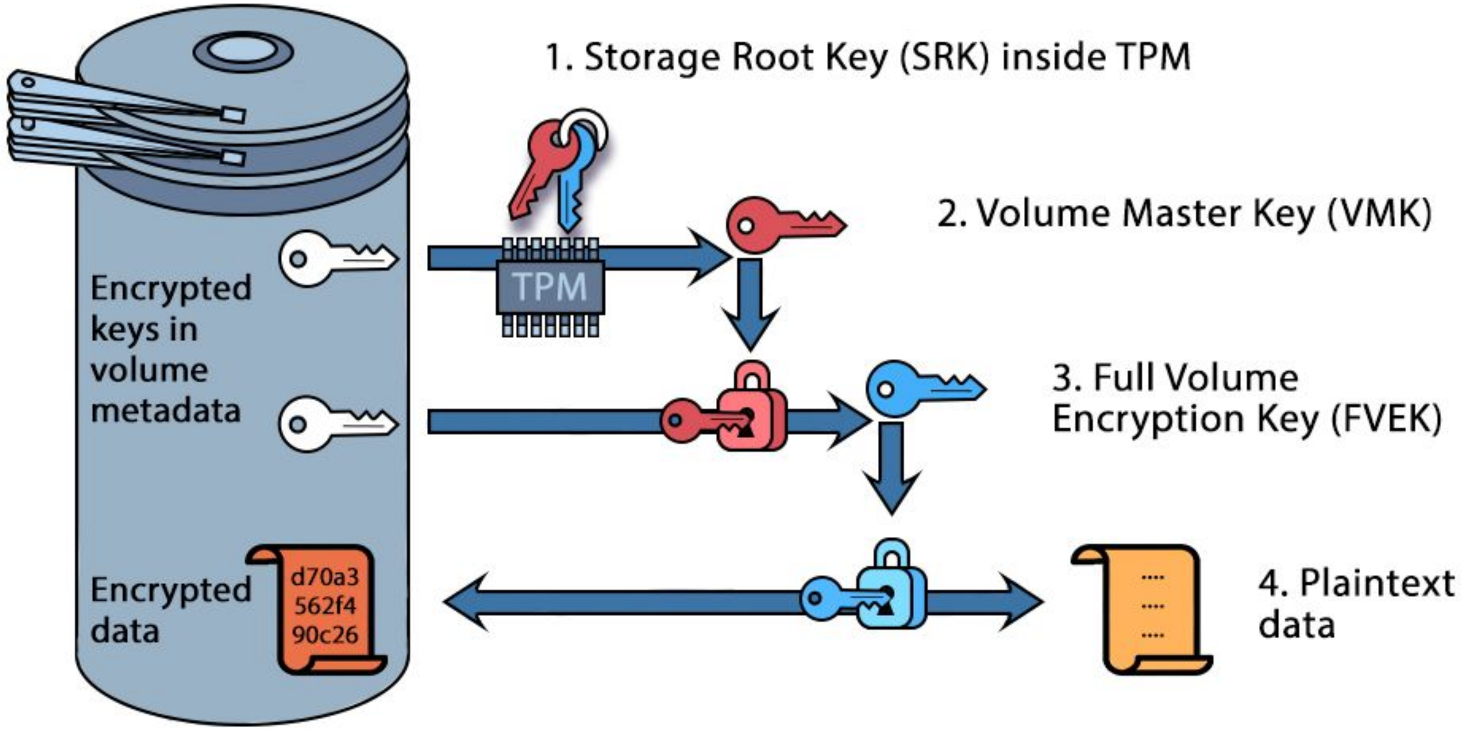


Bitlocker and key protectors

TPM only (the most used protector type on portable devices such as notebooks, Windows tablets and two-in-ones):

- the VMK will be decrypted with a storage root key (SRK) that is stored in the TPM module and only releases if the system passes the **Secure Boot check**
    - Secure Boot check enforces that the computer runs with the **original OS** in its **original configuration**
- a fully encrypted BitLocker volume will be **automatically mounted** and **unlocked** during the **Windows boot process**, long before the user signs in to the system with their Windows credentials

Bitlocker Keys with TPM


1. Storage Root Key (SRK) inside TPM
2. Volume Master Key (VMK)
3. Full Volume Encryption Key (FVEK)
4. Plaintext data
Encrypted keys in volume metadata
Encrypted data
d70a3
562f4
90c26
TPM

(https://blog.elcomsoft.com/2021/01/understanding-bitlocker-tpm-protection/)

Bitlocker and key protectors [2]

TPM + PIN

- the TPM module will only release the encryption key if the **PIN code** is correctly type during pre-boot phase
- even though the PIN code is short, entering the wrong PIN **several times** makes TPM panic and **block access** to the encryption key

BitLocker Encryption and threats

- the HD is **removed** from a computer
  - *password*: data are securely protected with a 128-bit encryption key (users requiring higher-level security can specify 256-bit encryption when setting up BitLocker)
- the **entire** computer is **stolen**
  - *TPM only*: the security of data depends on the strength of your Windows password
  - *TPM + PIN* is significantly more secure
- **Other users** on the same computer
  - If anyone can log in to your computer and access their account, the disk volume has been already decrypted: BitLocker does not protect against peer computer users
- **Malware**/ransomware and online **threats**
  - BitLocker does nothing to protect your data against malware, ransomware or online threats

Bitlocker: show status

```
PS C:\Windows\system32> cmd
BitLocker Drive Encryption: Configuration Tool
version 10.0.22000
Copyright (C) 2013 Microsoft Corporation. All
rights reserved.

Disk volumes that can be protected with
BitLocker Drive Encryption:
Volume C: []
[OS Volume]

    Size:                 470.03 GB
    BitLocker Version:    None
    Conversion Status:    Fully Decrypted
    Percentage Encrypted: 0.0%
    Encryption Method:    None
    Protection Status:    Protection Off
    Lock Status:          Unlocked
    Identification Field: None
    Key Protectors:       None Found
```

```
PS C:\Windows\system32> manage-bde.exe -status
BitLocker Drive Encryption: Configuration Tool
version 10.0.22000
Copyright (C) 2013 Microsoft Corporation. All
rights reserved.

Disk volumes that can be protected with
BitLocker Drive Encryption:
Volume C: []
[OS Volume]

    Size:                 469.34 GB
    BitLocker Version:    2.0
    Conversion Status:    Fully Encrypted
    Percentage Encrypted: 100.0%
    Encryption Method:    XTS-AES 128
    Protection Status:    Protection On
    Lock Status:          Unlocked
    Identification Field: Unknown
    Key Protectors:
        TPM
        Numerical Password
```

Breaking BitLocker Password

BitLocker poses a problem for forensic investigators, as all information on the drive will be encrypted, and therefore unreadable. Some methods for breaking BitLocker password are:

- **the RAM dump/hibernation file/page file attack**: this attack is universal, and works regardless of the type of protector. It **dumps** from the computer's volatile **memory** (and possibly in the page/hibernation file) the VMK that is loaded unencrypted while the volume is mounted
- **BitLocker recovery keys**: in many situations recovery keys are be **stored** in the user's **Microsoft Account**. Extracting those keys from their account allows instantly mounting or decrypting protected volumes regardless of the type of protector

Forensic acquisition: “the old mantra”

In making forensic acquisition, “*pulling the power plug*” has been considered the **safer option** to **preserve evidences** (e.g., not to lose temporary files or not to change the time stamps)

- considering TPM and Bitlocker: a live analysis should be performed to ensure that disk encryption is not in place

The right way to acquire a PC with a crypto container can be described with the following sentence: “*If it’s running, don’t turn it off. If it’s off, don’t turn it on*” (**encryption keys** can be extracted from **memory pages** or **hibernation files**).

Preserve evidences: Write Blockers

Write Blockers are tools used to **prevent data** from being **modified** or **deleted** during forensic investigations. They are essential for maintaining the **integrity** of the evidence.

- typically implemented by specialized hardware (but supporting specific interfaces and protocol versions)
- software solutions exist
  - Linux distributions (with a kernel-specific write-blocking capability in place), e.g.,Tsurugi Linux
  - Windows software, e.g., SAFE Block by ForensicSoft

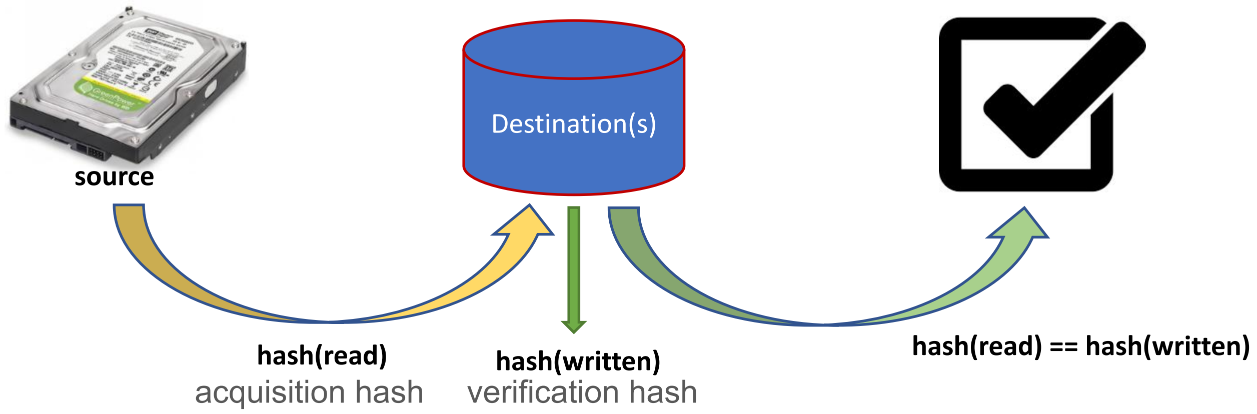
Physical acquisition

This type of acquisition involves creating a **bit-by-bit** copy (or **bitstream**) of the entire storage device

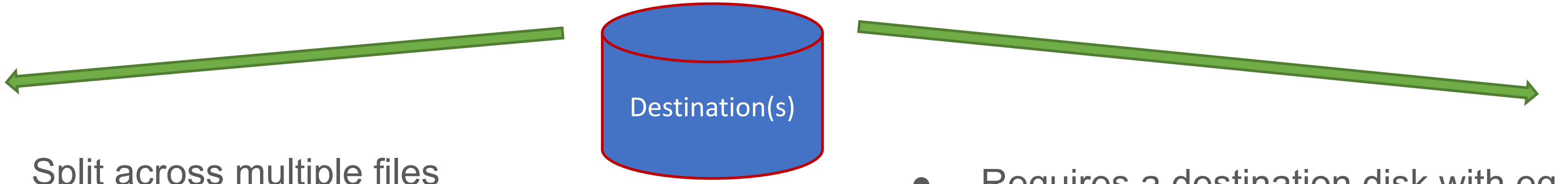
- it includes both **allocated** and **unallocated space** (may capture deleted files or other hidden data)
- should produce an exact copy of the **content** (you can not get an exact duplicate of the **container**)

Exact duplicate and Integrity

Assumed we are able to write block the source, how could we create an exact copy of the source?



Exact duplicate and Integrity: destination



File

- Split across multiple files
- Compression
- Encryption
- Metadata
- Simplified hash calculation
- A write blocker is not essential to access the file in read-only mode
- Multiple devices can be acquired on the same destination unit

Device

- Requires a destination disk with equal or greater capacity compared to the original
- Requires a destination disk for each original disk to be copied
- Requires wiping of the destination disk (to avoid cross-contamination)
- Requires that copies also be treated with a write blocker

Exact duplicate and Integrity: hashes

- Hashing and Documentation
  - Once hashes are calculated and verified, the entire process, including sources, destinations, and hash results, must be documented
- What Does a Hash Represent?
  - A certificate of the forensic imaging process
  - The integrity of the forensic image itself
- Single Hash for 2TB of Data?
  - A single-bit change (e.g., corruption) invalidates the entire hash
  - Solution: Use **piecewise hashing** – generate hashes for fixed-size segments of the file to improve integrity verification

Logical acquisition

A **logical image** of a device captures all (or a part of) **data visible to the user**.

- sometime physical acquisition is not feasible
  - data are too large to be copied
  - critical systems
  - …
- need to be forensically sound
  - requires reporting the reason why, actions, and hashes

Toolset and examples

Loop device

The loop device is a **block device** that maps its data blocks not to a physical device (e.g., an hard disk), but to the **blocks** of a **regular file** in a filesystem.

- useful to access a forensic image
- read/only can be forced
- the offset parameter could be useful to directly access a volume
- (can be used to simulate a block device to be acquired)

```
losetup - set up and control loop
devices

-a show status of all loop devices
-d detach the file or device associated
with the specified loop device
-f find the first unused loop device.
-o the data start is moved offset bytes
into the specified file or device…
-r setup read-only loop device
…


# losetup –a
# losetup -r /dev/loop0 [srcfile]
```

`dd`

`dd` is the precursor of all acquisition tools, allowing for the acquisition of data **bit by bit** in **raw** format.

- expect to find it in any *nix distribution
- critical options needed to perform a bitstream copy are usually available, even if not all distributions have every option
  - `if`, `of`, `bs`, `conv` (see later)
- an example disk-to-file acquisition of `/dev/sda` to the file `image.dd` with a block size of 512 (read and write up to 512 bytes at a time)
  - `dd if=/dev/sda of=/mnt/dest/image.dd bs=512`

dd: example

Simulate a block device with losetup (compressed image is available at https://github.com/enricorusso/DF_Exs/raw/main/acquisition/image.dd.gz)

- find a unused loop device, e.g., `/dev/loop1` (`losetup -f`)
- setup the block device: sudo losetup -r /dev/loop1 [image file]

  -
    ```
    sudo dmesg
    [84058.342422] loop1: detected capacity change from 0 to 2033664
    ```
    (2033664 sectors of 512 bytes each)

  -
    ```
    fdisk -l /dev/loop1

    Disk /dev/loop1: 993 MiB, 1041235968 bytes, 2033664 sectors
    Units: sectors of 1 * 512 = 512 bytes
    Sector size (logical/physical): 512 bytes / 512 bytes
    I/O size (minimum/optimal): 512 bytes / 512 bytes
    Disklabel type: dos
    Disk identifier: 0x0099b146

    Device       Boot Start     End Sectors  Size Id Type
    /dev/loop1p1 *       32 2033663 2033632  993M  c W95 FAT32 (LBA)
    ```

```
md5sum image.dd
446144a4af914d7e55603b6042f20db1  image.dd

sha1sum image.dd
99540f5aaa170afbab722729e980fd6dc34ff323
image.dd

md5sum /dev/loop1
446144a4af914d7e55603b6042f20db1
/dev/loop1

sha1sum /dev/loop1
99540f5aaa170afbab722729e980fd6dc34ff323
/dev/loop1
```

*use `partx -a /dev/loop1` to create the device files for each partition (e.g., `/dev/loop1p1`)

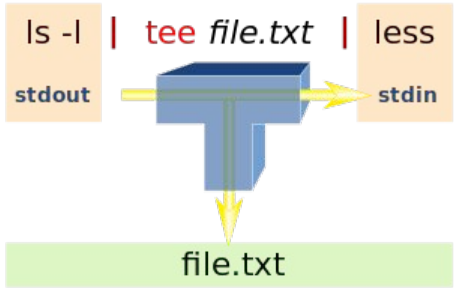
dd: example (with hashes)

- The dd tool does not calculate any hash
- A possible procedure could be
    - Calculate the source hash(es)
    - Make the acquisition
    - Calculate the destination hash(es)
- A slightly faster version

    -
      ```
      sudo dd if=/dev/loop1 bs=512 | tee image.dd | hashdeep -c md5,sha1 > image.src_hash (md5:
      446144a4af914d7e55603b6042f20db1, acquisition hash)
      ```

    -
      ```
      md5sum image.dd
      446144a4af914d7e55603b6042f20db1  image.dd (verification hash)

      (apt install hashdeep)
      ```



A faulty disk

- We create a logical "faulty" device with the command `dmsetup`*

  ```
  sudo dmsetup create bad_disk << EOF
  0 8       linear /dev/loop1 0
  8 8       error
  16 2033648 linear /dev/loop1 16
  EOF
  ```

  It maps all the sectors of the original disk except for 8 ones starting from the 9th which are marked as faulty (`sudo badblocks -b 512 -v /dev/mapper/bad_disk`).

- Set readahead to 0 (disable cache and force to fetch data directly from the disk), check block device size

  ```
  sudo blockdev --setra 0 /dev/mapper/bad_disk
  sudo blockdev --getsz /dev/mapper/bad_disk (2033664)
  ```

*https://mbroz.fedorapeople.org/talks/DeviceMapperBasics/dm.pdf

A faulty disk: acquisition with dd

- Acquire the faulty disk with dd

  ```
  sudo dd if=/dev/mapper/bad_disk of=bad.dd bs=512
  dd: error reading '/dev/mapper/bad_disk': Input/output error
  8+0 records in
  8+0 records out
  4096 bytes (4,1 kB, 4,0 KiB) copied, 0,001521 s, 2,7 MB/s
  ```

  The acquisition stops when the first bad sector is read.

- The generally accept behavior for dealing with a bad sector is to log its address and **write 0s for the data that could not be read** (writing 0s keeps the other data in its correct location, i.e., the sectors are aligned)
  - set the option `conv=sync,noerror` in dd

A faulty disk: acquisition with dd

- execut dd with `sync`, `noerror`

  ```
  sudo dd if=/dev/mapper/bad_disk bs=512 conv=sync,noerror | tee bad.dd | hashdeep -c
  md5,sha1 > bad_image.src_hash
  [...]
  2033656+8 records in
  2033664+0 records out
  1041235968 bytes (1,0 GB, 993 MiB) copied, 18,0024 s, 57,8 MB/s

  (446144a4af914d7e55603b6042f20db1,99540f5aaa170afbab722729e980fd6dc34ff323)
  ```

  8 sectors were not read but replaced with 0.

- `hashdeep -c md5,sha1 bad.dd`

  ```
  1041235968,446144a4af914d7e55603b6042f20db1,99540f5aaa170afbab722729e980fd6dc34ff323,/tmp
  /bad.dd
  ```

`dc3dd`

A patched version of GNU dd with added features for computer forensics, developed at the DoD Cyber Crime Center by Jesse Kornblum. It supports

- On the fly hashing with multiple algorithms (MD5, SHA-1, SHA-256, and SHA-512) with variable sized piecewise hashing
- writing errors directly to a file
- combined error log, pattern wiping, verify mode, progress reports, split output

`dc3dd`: example

An example with the faulty disk.

```
sudo dc3dd if=/dev/mapper/bad_disk of=bad.dd ssz=512 log=image.log hlog=hash.log hash=md5 hash=sha1

dc3dd 7.2.646 started at 2025-03-06 08:13:03 +0100
compiled options:
command line: dc3dd if=/dev/mapper/bad_disk of=bad.dd ssz=512 log=image.log hlog=hash.log hash=md5 hash=sha1
device size: 2033664 sectors (probed),    1,041,235,968 bytes
sector size: 512 bytes (set)
trying to recover sector 16
[!!] 8 occurrences while reading `/dev/mapper/bad_disk' from sector 8 to sector 15 : Input/output error
  1041235968 bytes ( 993 M ) copied ( 100% ),   19 s, 51 M/s

input results for device `/dev/mapper/bad_disk':
   2033664 sectors in
   8 bad sectors replaced by zeros
   8f07472b6cce371bb2af0b78f19287ee (md5)
   ac68bd94e90adb64c59838edd9b97b90a0c7e042 (sha1)

output results for file `bad.dd':
   2033664 sectors out

dc3dd completed at 2025-03-06 08:13:22 +0100
```

Image file formats

- the output from dd acquisition is a raw image
  - it contains only the data from the source device
  - all the descriptive data about the acquisition (e.g., hashes values, dates, or times) need to be saved in a separate file
- an embedded image contains data from the source device and additional descriptive data (metadata)
  - Expert Witness Format
  - Advanced Forensic Format

Expert Witness Format

- de facto standard for forensic analysis but it has a proprietary format (compatibility is achieved through reverse engineering)
- supported by mainstream open source and commercial analysis software
- include metadata (although limited) in the acquired image:
    - date/time of acquisition
    - examiner name
    - extra notes
    - password
    - MD5 hash of the entire image
- supports image compression
- search within the acquired image
- images can be split and mounted on-the-fly

Expert Witness Format: `libewf`

Joachim Metz (Google) created the `libewf` project, open source (https://github.com/libyal/libewf, `apt install ewf-tools`). It provides a library and set of tools to manage the ewf format.

- `ewfacquire`: reads storage media data from devices and write files to EWF files.
- `ewfexport`: exports storage media data in EWF files to (split) RAW format or a specific version of EWF files.
- `ewfinfo`: shows the metadata in EWF files.
- `ewfmount`: FUSE mounts EWF files.
- `ewfrecover`: special variant of ewfexport to create a new set of EWF files from a corrupt set.
- `ewfverify`: verifies the storage media data in EWF files.

Expert Witness Format: ewfacquire (example)

Acquire the faulty disk (answering the questions)

```
ewfacquire -b 16 -g 1 -r 0 -d sha1 -t bad /dev/mapper/bad_disk

The following acquiry parameters were provided:
Image path and filename:       mycase.E01
Case number:                          1
Description:                          My test case
Evidence number:               1234
Examiner name:                        Enrico Russo
Notes:
Media type:                          fixed disk
Is physical:                         yes
EWF file format:                     EnCase 6 (.E01)
Compression method:                  deflate
Compression level:                   best
Acquiry start offset:                0
Number of bytes to acquire:          993 MiB (1041235968
bytes)
Evidence segment file size:          1.4 GiB (1572864000
bytes)
Bytes per sector:                    512
Block size:                          16 sectors
Error granularity:                   1 sectors
Retries on read error:               16
Zero sectors on read error:          no

Continue acquiry with these values (yes, no) [yes]:
```

```
Acquiry started at: Mar 06, 2025 08:24:20
This could take a while.

Status: at 27%.
        acquired 275 MiB (288358400 bytes) of total 993 MiB
(1041235968 bytes).
        completion in 10 second(s) with 70 MiB/s (74373997
bytes/second).

[...]

Acquiry completed at: Mar 06, 2025 08:38:04

Written: 993 MiB (1041238048 bytes) in 25 second(s) with 39
MiB/s (41649521 bytes/second).
Errors reading device:
        total number: 1
        at sector(s): 0 - 16 number: 16 (offset: 0x00000000 of
size: 8192)

MD5 hash calculated over data:
e436e37a1cb881844e01ff44ccae61c0
SHA1 hash calculated over data:
7da3378f09c3b0f8ec1f73a01e50eed3cf4e2082
ewfacquire: SUCCESS
```

*test the hash mismatch with `ewfacquire -b 16 -g 1 -r 0 -d sha1 -t bad /dev/loop1` and
`dc3dd if=/dev/mapper/bad_disk of=bad.dd ssz=8192 log=image.log hlog=hash.log hash=md5 hash=sha1`

Expert Witness Format: ewinfo (example)

```
ewfinfo mycase.E01

ewfinfo 20140807

Acquiry information
	Acquisition date:	Thu Mar  6
08:37:52 2025
	System date:		Thu Mar  6
08:37:52 2025
	Operating system used:Linux
	Software version used:20140807
	Password:	N/A

EWF information
	File format:		EnCase 6
	Sectors per chunk:	16
	Error granularity:	1
	Compression method:	deflate
	Compression level:	best compression
```

```
Media information
	Media type:	fixed disk
	Is physical:		yes
	Bytes per sector:	512
	Number of sectors:	2033664
	Media size:	993 MiB (1041235968
bytes)

Digest hash information
	MD5:
e436e37a1cb881844e01ff44ccae61c0
	SHA1:
7da3378f09c3b0f8ec1f73a01e50eed3cf4e2082

Read errors during acquiry
	total number: 1
	at sector(s): 0 - 15 number: 16
```

Advanced Forensic Format

Open Source format developed by Dr. Simson L. Garfinkel

- Provide compressed or uncompressed image files
- No size restriction for disk-to-image files
- Provide space in the image file or segmented files for metadata (unlimited number)
- Digital signatures
- Encryption with decryption on-the-fly
- No patents

Still lacks wide adoption (software available at https://github.com/sshock/AFFLIBv3).

Guymager

Guymager is a graphical (Qt-based) forensic imager. It is capable of producing image files in EWF, AFF and dd format (apt install guymager).

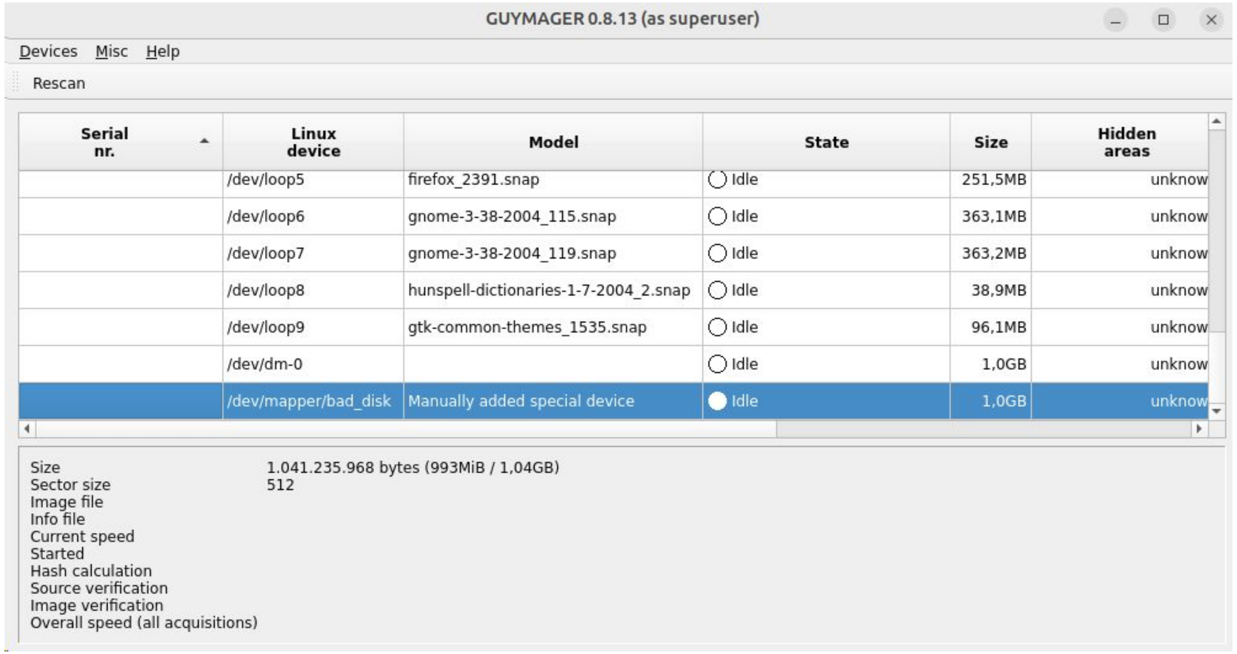
AFF is disabled by default (see `/etc/guymager/guymager.cfg`)

```
REM AffEnabled            Simson Garfinkel, the inventor of the AFF format,
recommends not to use AFF any longer.
REM                       Therefore, this switch has been introduced and it
is 'false' by default. You might use EWF
REM                       instead.
REM                       Switch AffEnabled on in case you need to generate
AFF images.
```

It uses an internal library for managing the EWF .

Guymager

- Add a new special device for the faulty disk (menu *Devices* > *Add special device*), right click on the device then select *Acquire image*.

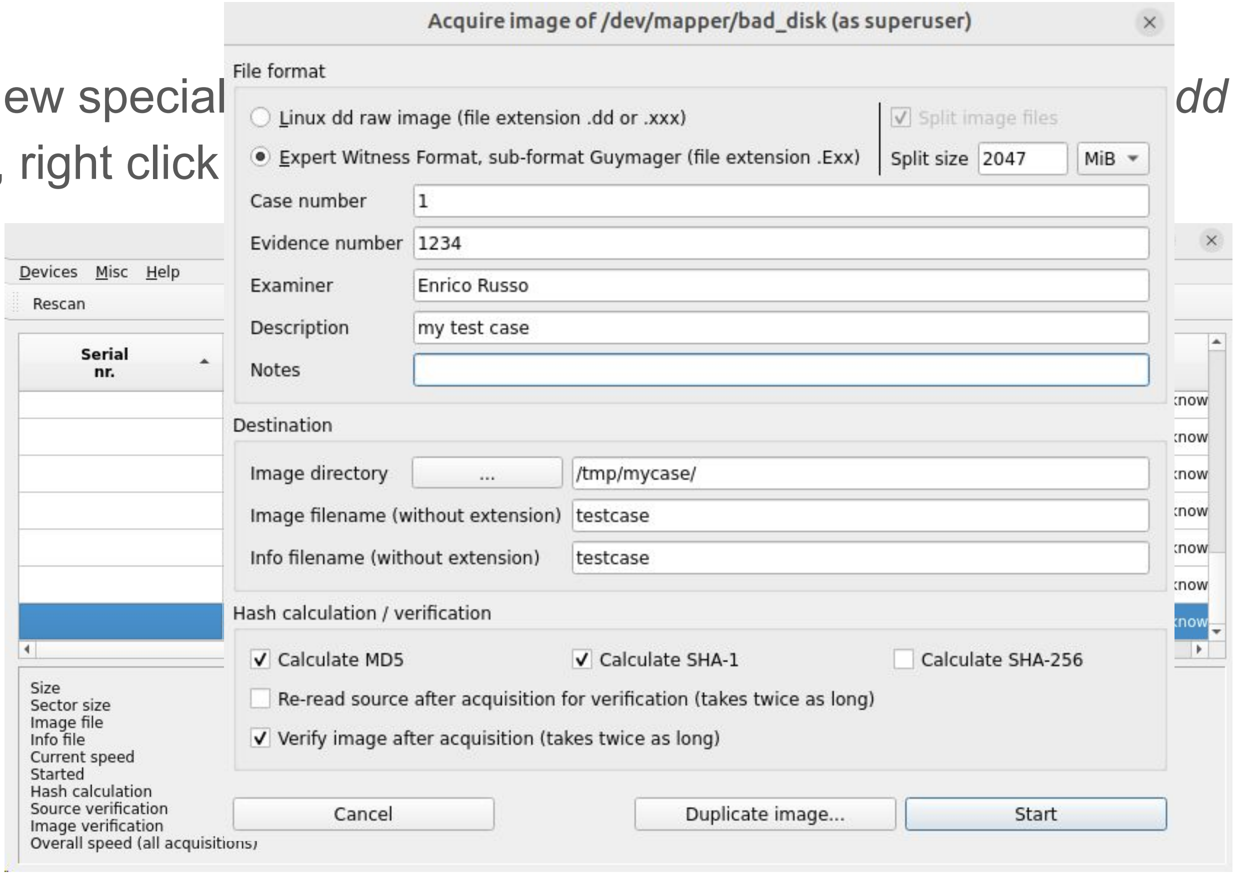


Serial nr.	Linux device	Model	State	Size	Hidden areas
	/dev/loop5	firefox_2391.snap	◯ Idle	251,5MB	unknow
	/dev/loop6	gnome-3-38-2004_115.snap	◯ Idle	363,1MB	unknow
	/dev/loop7	gnome-3-38-2004_119.snap	◯ Idle	363,2MB	unknow
	/dev/loop8	hunspell-dictionaries-1-7-2004_2.snap	◯ Idle	38,9MB	unknow
	/dev/loop9	gtk-common-themes_1535.snap	◯ Idle	96,1MB	unknow
	/dev/dm-0		◯ Idle	1,0GB	unknow
	/dev/mapper/bad_disk	Manually added special device	● Idle	1,0GB	unknow

```
Size                            1.041.235.968 bytes (993MiB / 1,04GB)
Sector size                     512
Image file
Info file
Current speed
Started
Hash calculation
Source verification
Image verification
Overall speed (all acquisitions)
```

Guymager

- Add a new special *dd special device*), right click



Acquire image of /dev/mapper/bad_disk (as superuser)

File format

- ○ Linux dd raw image (file extension .dd or .xxx)
- ● Expert Witness Format, sub-format Guymager (file extension .Exx)

[✓] Split image files

Split size: 2047 MiB

Case number	1
Evidence number	1234
Examiner	Enrico Russo
Description	my test case
Notes	

Destination

Image directory	/tmp/mycase/
Image filename (without extension)	testcase
Info filename (without extension)	testcase

Hash calculation / verification

- [✓] Calculate MD5
- [✓] Calculate SHA-1
- [ ] Calculate SHA-256
- [ ] Re-read source after acquisition for verification (takes twice as long)
- [✓] Verify image after acquisition (takes twice as long)

Windows: FTK Imager

- it’s possible to make acquisition using the Windows OS too
  - assumed a proper write blocking
- some of the previous linux tools have a Windows versions
- one of the most used tools is the *free* Exterro (AccessData) FTK Imager (https://www.exterro.com/ftk-product-downloads)

  FTK® Imager is a data preview and imaging tool used to acquire data (evidence) in a forensically sound manner by creating copies of data without making changes to the original evidence

FTK Imager

- uses a proprietary format
- capable of doing logical acquisition of selected files (e.g., Windows shares)
- can acquire memory too
- can be useful to explorer the data acquired
    - supports different file systems (e.g., ext, apfs, etc.)
    - supports different input formats (e.g., vmdk, dmg, etc.)
- It’s able to mount a forensic image
    - Windows only supports FAT family and NTFS