

Digital Forensics Principles: Never Touch the Original

Fri, Sep 4, 10:00 AM EDT · <https://scottslab.io/posts/digital-forensics-principles>



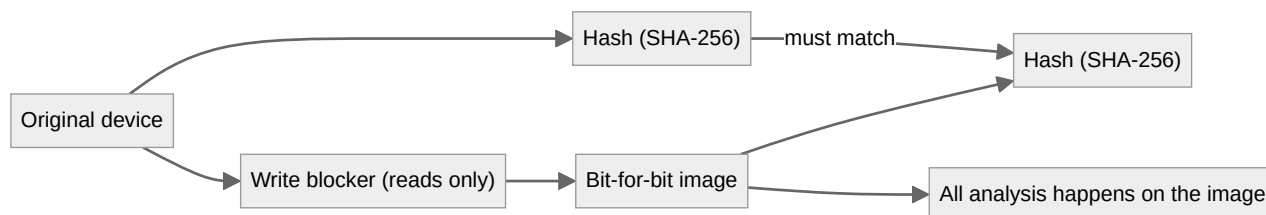
TL;DR

The first rule of digital forensics is you never alter the evidence. You work from a bit-for-bit image, not the original media, with a write blocker between your system and the device so you can't accidentally write to it. Two standards codify this: ISO/IEC 27037 for identification, collection, acquisition and preservation of digital evidence, and NIST SP 800-86 for integrating forensic technique into incident response. A hash proves the image matches the source. SHA-256 is standard now, because both MD5 and SHA-1 have practical collision attacks. Collect in RFC 3227's order of volatility, fastest-vanishing first, and record the device's clock offset at capture.

Everything in digital forensics descends from one rule: never alter the evidence. The moment you modify the original media, you've handed opposing counsel a reason to throw it all out. So you don't work on the original. You take a bit-for-bit image and work from the copy, and you put a write blocker between your forensic system and the device: hardware (or software) that physically allows reads but blocks writes, so even the OS's habit of quietly touching a disk can't change what's there.

This isn't folklore, it's standardized. **ISO/IEC 27037:2012**, "Guidelines for identification, collection, acquisition and preservation of digital evidence," names those four activities as the

discipline's core sequence, and the "never alter" rule is baked into all four: identify it without touching it, collect and acquire a copy, preserve the original untouched. **NIST SP 800-86**, "Guide to Integrating Forensic Techniques into Incident Response," is the companion piece most incident responders actually reach for, because it ties the forensic process into the incident-handling workflow rather than treating forensics as something that only happens after the fact.



Digital Forensics Principles: Never Touch the Original

Hashing, and why the algorithm choice isn't trivia

Proving you didn't alter the evidence is where hashing comes in. You hash the original and the image; if the hashes match, the copy is provably identical, and because a single changed character produces a completely different hash, any later tampering is instantly detectable.

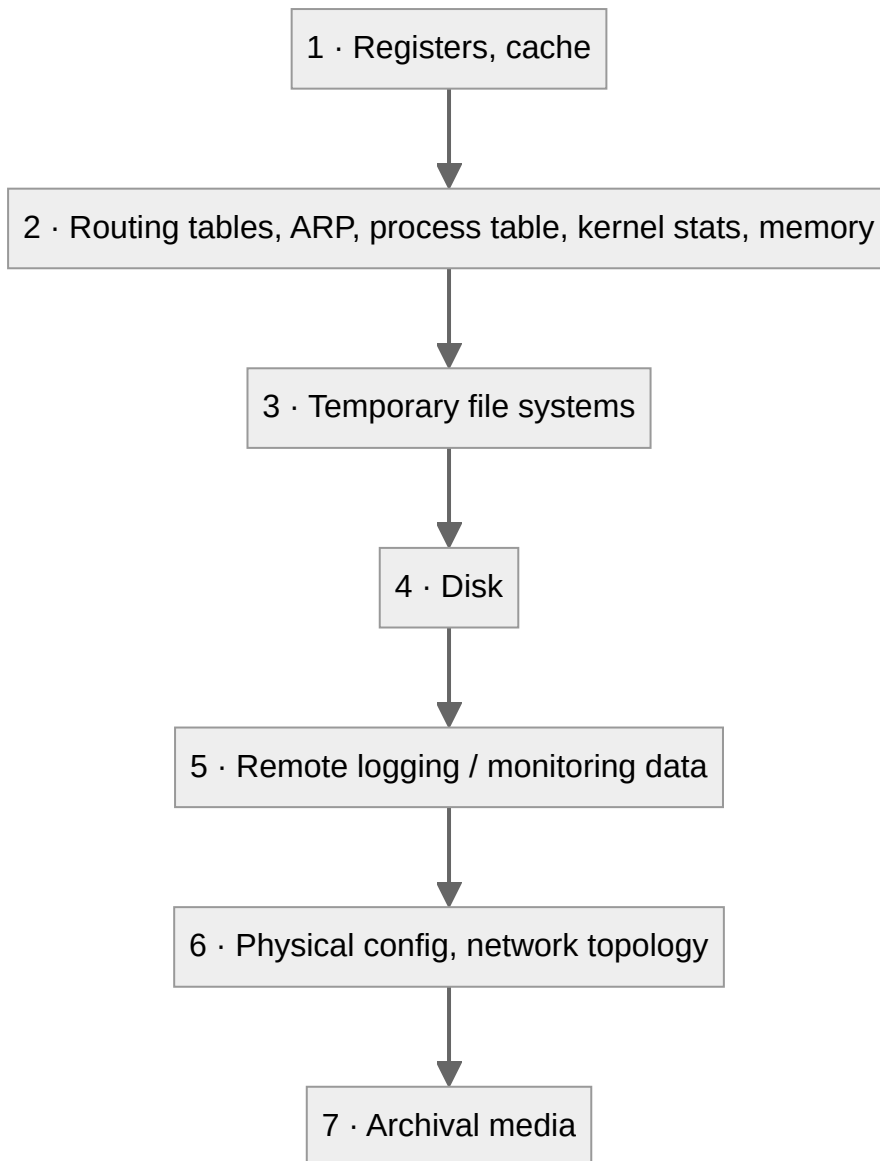
The algorithm matters more than it looks. MD5 and SHA-1 were the field's defaults for years, and both now have practical collision attacks. The one that mattered was "SHAttered," published by Google and CWI Amsterdam in February 2017, the first public demonstration of two different files sharing a SHA-1 hash. That's a real problem for forensic integrity in theory: a hash algorithm with known collisions can't rule out that two different pieces of evidence were engineered to match. In practice, most forensic hashing is defending against *accidental* change, not an adversary crafting a collision against your specific image. That's why some legacy tooling still runs MD5 alongside a stronger hash out of habit. But there's no good reason to lean on a broken algorithm when SHA-256 costs nothing extra, and it's the standard now for exactly that reason. A digital signature goes one step further, adding non-repudiation on top of the hash: not just "this wasn't changed" but "this came from me, and I can't credibly deny it."

The hash isn't just internal hygiene, either. It's what makes an image or a system-generated record legally self-authenticating without a witness in most federal cases (Federal Rule of Evidence 902(14), covered in the evidence-admissibility post in this series). The forensic discipline and the legal rule that lets it work are the same idea from two directions.

Order of volatility

Timing and sequence matter as much as integrity. **RFC 3227**, the IETF's guidelines for evidence collection, defines the canonical order for a live system: proceed from the volatile to the less volatile. Registers and cache first; then routing tables, ARP cache, process table, kernel statistics,

and memory; then temporary file systems; then the disk itself; then remote logging and monitoring data relevant to the system; then physical configuration and network topology; archival media last. Reboot before you capture memory and it's gone forever. No image recovers it afterward.



Order of volatility

Record the time offset at capture too: the device's clock versus a reliable source. A machine that's twelve minutes fast will otherwise scramble your timeline against everyone else's logs. RFC 3227 makes this an explicit collection step for the same reason: a timeline built on unreconciled clocks isn't a timeline, it's a coincidence.

Imaging vs. live acquisition

The write-blocker workflow above assumes you can power the system down and image a static disk: a "dead-box" acquisition. That's the gold standard for integrity, but it throws away everything volatile: memory contents, running processes, open network connections, encryption keys held only

in RAM. If the case turns on any of that, you need a **live acquisition** first. That means capturing memory and process state from the running system, accepting that the act of capturing changes the system in small, documented ways, before you ever pull the plug. Neither approach is wrong; they answer different questions. The decision of which to run first is exactly what the order-of-volatility sequence above is for: it tells you what you lose if you image before you capture.

The quiet goldmine: metadata

File metadata deserves its own line. Creation and modification timestamps, and EXIF data baked into photos (camera model, and often GPS coordinates), are frequently the evidence itself: the "anonymous" photo that carries the location it was taken, the document whose modification time contradicts the story someone told you. Capture it, because it's exactly the kind of detail that doesn't survive careless handling of the original, which is the whole reason rule one exists in the first place.