Hardware Forensic Suite: A Portable Multi-Platform Digital Evidence Acquisition Framework

[1]Prof. Swapnil Wani, [2]Ashish Mishra, [3]Sameer Yadav, [4]Siddhesh Tambe, [5]Soham Desai
[1]UG Guide, [2,3,4,5]UG Scholar, Department of Computer Science and Engineering (IoT & CSIBCT), SIES Graduate School of Technology, Nerul, Navi Mumbai, India.
[1]swapnilw@sies.edu.in, [2]ashishjmiot122@gst.sies.edu.in, [3]sameerayiot122@gst.sies.edu.in, [4]siddheshptiot122@gst.sies.edu.in, [5]sohamddiot122@gst.sies.edu.in

***Abstract*—When a digital forensic investigation kicks off, investigators typically find themselves juggling three or four separate tools just to cover the basics — one for memory, another for disk imaging, a third for network capture. Each runs independently, demands manual setup, and hands off nothing to the next. The result is slow, error-prone work that can compromise the very evidence being collected. This paper introduces the Hardware Forensic Suite (HFS), a portable acquisition framework built to fix exactly that. A forensic USB device is connected to the target machine; a Universal Loader detects the host OS and silently launches the right agent for that environment. From there, three modules take over in sequence — memory first, then disk, then network — without the investigator touching a thing. SHA-256 hashing and write-blocking run throughout, and every acquisition event gets chained into a tamper-evident log for legal use. We built and tested all three modules independently on Ubuntu 22.04 and Windows 10. Both environments passed cleanly, confirming that the architecture holds up before we push to full integration.**

***Index Terms*—digital forensics, memory acquisition, disk imaging, network forensics, write blocker, chain of custody, portable forensics, forensic automation**

I. INTRODUCTION

Digital forensics sits at a strange intersection — it demands both technical precision and speed, often under pressure, in environments where a wrong move can destroy evidence or render it inadmissible. Whether the trigger is a ransomware incident, corporate fraud, or a criminal investigation, the investigator needs to capture memory, disk, and network state before the system changes. Every minute counts.

The reality on the ground, though, is messy. Most investigators work with a collection of standalone tools that were never designed to talk to each other. LiME or WinPmem for memory, dd or FTK Imager for disk, Wireshark for network — each launched separately, each requiring configuration, each producing output the others know nothing about. On top of that, many tools are tied to a specific OS, so switching from a Linux box to a Windows machine mid-investigation means swapping setups entirely. The chain of custody often gets documented manually, after the fact, which leaves room for gaps that defence counsel will happily exploit.

We built the Hardware Forensic Suite to address these issues directly. The idea is simple: plug in a USB device, and the acquisition starts on its own. A Universal Loader figures out what OS it is dealing with and launches the right agent. That agent runs memory capture, disk imaging, and network forensics in the correct order, hashing and logging every step as it goes. No manual sequencing. No platform-specific setup. No post-acquisition documentation scramble.

The system currently targets Windows and Linux. The modular design means adding macOS or embedded platforms later is a matter of writing a new agent, not redesigning the core.

Contributions of this paper:

- A unified acquisition framework that handles memory, disk, and network evidence collection under one automated workflow, removing the need to coordinate between separate tools.
- A Universal Loader and Master Agent design that detects the host platform and runs the right acquisition stack without installing anything on the target machine.
- Working implementations of all three acquisition modules, tested independently on Linux and Windows with verified integrity results.
- Built-in write-blocking, SHA-256 hashing at every module boundary, and hash-chained logging that produces a chain-of-custody record as a natural byproduct of acquisition rather than a separate step.

 DOI : 10.35291/2454-9150.2026.0033

II. LITERATURE SURVEY

Foundations of data acquisition. The starting point for understanding what forensic acquisition actually requires is Jaganathan's work [1], which lays out the procedural and technical conditions for evidence to be considered sound. Abdallah [2] then situates this within the broader DAQ landscape, comparing hardware and software approaches. Raghavan [4] gives a useful bird's-eye view of where digital forensics research stands, covering both what has been solved and what hasn't. Fakhouri [5] and Mohay [6] are worth reading together — between them they cover most of the recurring headaches: tool fragmentation, evidence that vanishes before it can be captured, and the difficulty of reproducing acquisition results consistently.

Forensic tools and artifact sources. Ghazinour [3] took the time to actually compare the major forensic tools rather than just list them, and the gaps he identifies are part of what motivated the design choices in this work. Shah [7] makes the case that USB remnant data is genuinely useful evidence — something practitioners sometimes overlook. Cusack [8] makes a similar argument for network routers, which tend to hold connection logs and ARP tables that survive after a host is shut down. Zhang [9] demonstrated that pulling RAM over FireWire works without leaving a footprint on the target; that approach informed how we thought about the memory module's non-intrusive design.

Automation and formal frameworks. Korol's FEAR framework [10] is an attempt to give forensic processes a proper formal structure — something the field has needed for a while. Michelet [11] addresses the question of what automation even means in a forensic context, which turns out to be less obvious than it sounds. Holt [13] actually built and tested an automated forensic toolkit for incident response, so his findings carry real practical weight. Mohammed and colleagues [12], [15], [16] have published extensively on making sense of heterogeneous forensic data — work that becomes relevant once you are pulling artifacts from multiple modules simultaneously.

Portability, scalability, and emerging challenges. Yudha's Raspberry Pi-based imager [14] is the closest prior work to what we built — a compact, field-ready device that does not depend on the target's environment. Quick and Choo [17] document how fast the data volumes are growing and what that means for tools that were not designed to scale. Montasari [18] and Shin [19] look ahead to cloud and IoT contexts where the acquisition problem gets considerably harder. Reddy [20] rounds this out with a usability angle that is easy to dismiss but matters enormously in real investigations — a tool that confuses under pressure is a liability.

What the literature makes clear is that the pieces exist but nobody has assembled them. Individual tools work, automation is possible, portable hardware has been demonstrated — yet the field still largely operates with disconnected tools, manual logging, and no unified pipeline. That gap is what this work targets.

III. RESEARCH GAPS

The core problem is that forensic acquisition has never been treated as a single pipeline problem. Memory tools capture RAM; disk tools copy sectors; network tools sniff packets — and that is where the integration stops. An investigator working a live incident has to run these sequentially, by hand, switching tools between each step [3], [4]. Every context switch is an opportunity to make a mistake, and every minute spent configuring the next tool is a minute during which volatile data is disappearing.

Platform dependency makes this worse. A tool that works on Windows often needs significant rework to run on Linux, and vice versa. In field investigations where the target machine's OS is unknown until you walk in the door, this is a serious operational problem [6], [18]. Most existing tools simply assume a known environment.

Automation barely exists in practice. The typical forensic pipeline is: decide what to capture, launch the right tool, wait, verify the output by hand, document, move to the next step. Every one of those transitions is manual [11], [13]. Hash verification and chain-of-custody logging are usually afterthoughts — done once the acquisition is complete, if they are done at all [1], [15]. That is precisely where documentation gaps appear, and those gaps matter when evidence reaches a courtroom.

Portability is also an unsolved problem at the system level. Most frameworks require at least some software to be present on or installed onto the target — which is not always possible and always risks altering the system state [14], [5]. Meanwhile, the sheer volume of forensic data is growing fast enough that tools designed for yesterday's cases are already struggling [17], [16].

The Hardware Forensic Suite was designed with all of these gaps in view. The Universal Loader handles unknown environments. The Master Agent sequences everything automatically. Hashing and logging happen inline. Nothing gets installed on the target [11], [13], [14].

IV. SYSTEM DESIGN

A. System Architecture Overview

The HFS is built around four components that hand off to each other in a fixed sequence. Rather than a monolithic application, it is deliberately layered so that each piece can be swapped or extended independently:

- Universal Loader: This is the first thing that runs when the device is connected. It probes the host environment, figures out the OS and architecture, and hands control to the right Master Agent. No user input needed, nothing gets written to the target.
- Platform-Specific Master Agent: Think of this as the coordinator. It spins up the acquisition directory on the forensic device, fires each module in the right order, and

 DOI : 10.35291/2454-9150.2026.0033

keeps the log running throughout. If a module fails, it records that too.

- Memory Acquisition Module: Goes first, always. RAM is the most perishable evidence — running processes, open handles, encryption keys, network sockets — all of it disappears the moment the system restarts or even goes idle long enough. This module dumps it before anything else runs.
- Disk Imaging Module: Runs second. Every sector of the attached storage gets copied bit-for-bit, write protection enforced throughout. Deleted files, slack space, hidden partitions — it captures the full picture, not just what the filesystem shows.
- Network Forensics Module: Runs last. By this point the memory and disk are secured, so we can afford to take a moment to record the network state — active connections, routing tables, ARP and DNS caches, and a packet capture from each live interface.

The ordering is not arbitrary. Memory goes first because it is the most volatile. Disk goes second because a sector-level copy takes time and is not going anywhere. Network goes last because connection state changes gradually and the passive capture has no side effects on the target. This sequence is the right one for preserving evidence, and the Master Agent enforces it every time.

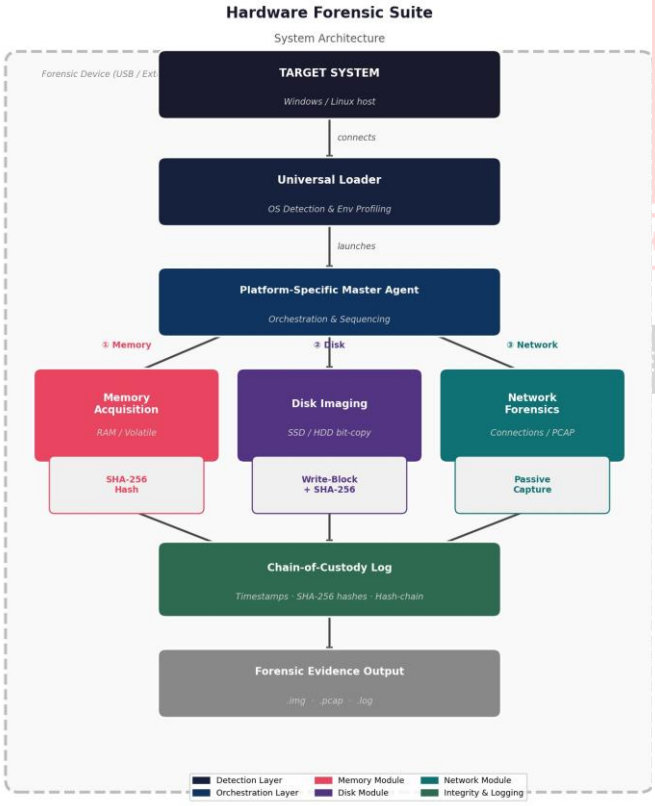

Fig. 1. Hardware Forensic Suite System Architecture

B. Universal Loader and OS Detection

The Universal Loader lives on the forensic device and runs the moment the device is connected — no double-clicking, no setup prompt. It queries the host environment for OSlevel identifiers, checks the architecture (32-bit vs 64-bit), and assesses what execution permissions are available. From those signals it picks the right Master Agent binary and launches it.

We formalise the selection process as a weighted confidence score over n environmental signals $\{s_1, s_2, \ldots, s_n\}$, where each signal carries a weight $w_i \in [0,1]$:

$$C_P = \frac{\sum_{i=1}^{n} w_i \cdot \mathbb{1}[s_i = P]}{\sum_{i=1}^{n} w_i} \quad (1)$$

The loader picks $P^{\wedge} = \text{argmax}_P\, C_P$ — whichever platform has the strongest signal agreement. This matters because edge cases exist: dual-boot machines, WSL environments, unusual kernel configurations. Voting across multiple signals produces a reliable answer even when any single signal would be ambiguous.

C. Master Agent and Module Orchestration

Once the Master Agent is running, it creates a timestamped output directory on the forensic device and begins driving the acquisition sequence. The ordering is derived from data volatility — the faster something disappears, the sooner it gets captured. Formally, each source d_i is assigned a volatility index $v_i \in \mathbb{R}^+$, and the execution order minimises expected evidence loss:

$$\pi^* = \text{argmax}^{X}_{\pi} \sum_{i=1}^{k} v_{\pi(i)} \cdot (k - i) \quad (2)$$

For our three-module system, this resolves to Memory $\succ$ Disk $\succ$ Network without ambiguity. After each module finishes, the Master Agent writes a log entry containing the module name, start and end timestamps, and the SHA-256 hash of everything captured. That entry then feeds into the next one via hash chaining.

D. Memory Acquisition Module

RAM is the most time-sensitive thing on a live system. Processes terminate, caches flush, encryption keys get zeroed — within minutes of any disruption, significant volatile data is simply gone. So this module runs first, reading physical memory through whatever interface is available: LiME on Linux, kernel-level interfaces on Windows. Kernel module loading is avoided where possible to keep the target's state as undisturbed as we can manage. The dump lands on the forensic device in a format that Volatility can ingest directly. Once complete, the image gets hashed:

$$H_{\text{mem}} = \text{SHA256}(M_{\text{raw}}) \quad (3)$$

H_{mem} is logged immediately, before the next module starts.

 DOI : 10.35291/2454-9150.2026.0033

E. Disk Imaging Module

A sector-level copy captures everything — not just what the filesystem reports, but deleted file remnants, unallocated space, slack space, and any hidden partitions. Before a single read happens, write protection goes up. On Linux this is done through block device controls; on Windows, through registry-level write protection. The goal is that the source drive looks identical at the end of imaging as it did at the start. We verify this by comparing per-sector CRC32 checksums between acquisition start (t_0) and end (t_1):

$$\Delta_{WB} = \bigoplus_{j=1}^{N} \mathrm{CRC32}(S_j^{t_0}) \oplus \mathrm{CRC32}(S_j^{t_1}) = 0 \quad (4)$$

Any deviation means something changed on the source — an integrity violation, logged and flagged immediately. The imaging itself uses dcfldd on Linux (we chose it over plain dd specifically for its inline hashing). We track how much of the disk we actually got:

$$\Phi = \frac{|\mathcal{S}_{\text{imaged}}|}{|\mathcal{S}_{\text{total}}|} \times 100\% \quad (5)$$

Then source and image hashes are compared:

$$H_{\text{source}} = H_{\text{image}} \Longleftrightarrow \text{acquisition verified} \quad (6)$$

A mismatch halts and flags. A match moves on to the next module.

F. Network Forensics Module

By the time the disk is imaged, the most fragile evidence is secured. The network module can afford to be methodical. It collects active TCP and UDP connections, routing tables, ARP and DNS caches, and runs a timed packet capture across every live interface. On Linux that means netstat, ss, arp, ip route, and tcpdump; on Windows, equivalent native commands and Wireshark CLI where available. The module never sends a single packet and never touches any network configuration.

On top of the raw capture, we compute the Shannon entropy of the observed protocol mix:

$$H_{\text{net}} = -\sum_{p \in P} \Pr(p) \log_2 \Pr(p) \quad (7)$$

An unexpectedly high H_{net} can be an early signal of data exfiltration or covert channel traffic — something worth flagging for the investigator before they even open the PCAP.

G. Integrity Verification and Chain of Custody

Every module produces a SHA-256 hash the moment it finishes. Those hashes, along with ISO 8601 timestamps, module identifiers, file paths, and acquisition status, go into a structured log on the forensic device. But the log itself also needs protection — an attacker who can alter the log can make a tampered image look clean. So we chain the entries:

$$L_i = \langle \tau_i, M_i, H_i, \mathrm{SHA256}(L_{i-1}) \rangle \quad (8)$$

Each entry carries the hash of the one before it. Verifying the full chain takes $O(n)$ time:

$$\text{Valid} \Longleftrightarrow \forall i \in \{1,\ldots,n\} : \mathrm{SHA256}(L_{i-1}) = L_i.\text{prev hash} \quad (9)$$

Modify any entry after the fact and every subsequent hash breaks. The chain-of-custody record is produced automatically during acquisition — not assembled retroactively from notes.

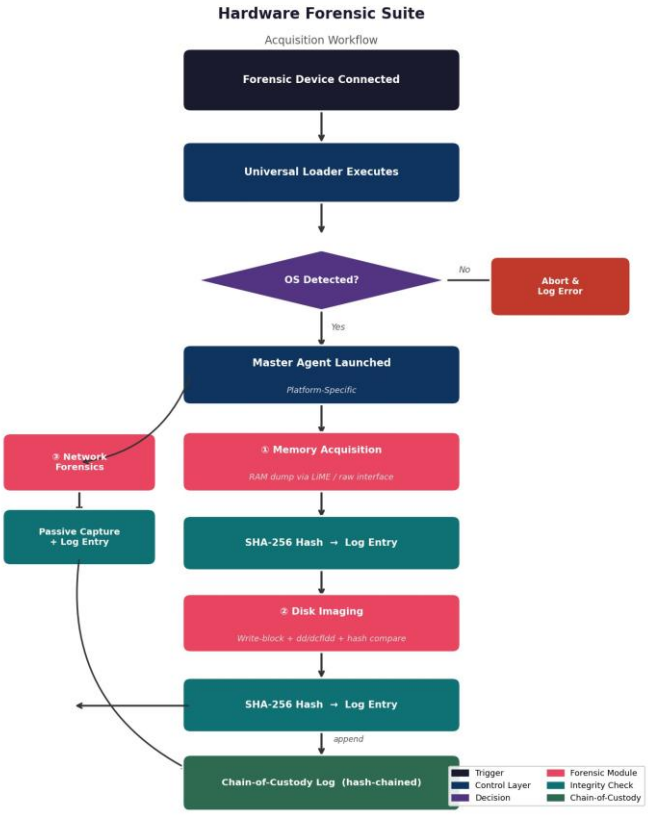

Fig. 2. Hardware Forensic Suite Acquisition Workflow

V. PROTOTYPE IMPLEMENTATION AND EVALUATION *A. Implementation Setup*

We ran the prototype on two machines: a laptop running Ubuntu 22.04 LTS and a desktop running Windows 10. For memory acquisition, LiME handled the Linux side and a compatible raw memory reader was used on Windows. Disk imaging went through dcfldd on Linux — we picked it over dd specifically because it computes hashes during imaging rather than after, saving a full re-read of the drive. Windows used a compatible sector-level imaging utility. Network collection ran through shell scripts on Linux and PowerShell on Windows, calling native tools in each case. SHA-256 hashing used sha256sum on Linux and CertUtil on Windows; both appended their output to the running acquisition log automatically.

 DOI : 10.35291/2454-9150.2026.0033

B. Evaluation Methodology

We evaluated each module on three questions: did it capture everything it was supposed to, did it disturb the target system noticeably, and did the hash verification hold across runs? Completeness was checked by comparing acquired images against a reference snapshot taken before each test. System impact was assessed by watching CPU usage, I/O wait, and process scheduling during acquisition — if a module was causing disruption, we would see it there. Hash consistency was confirmed by running each acquisition three times on the same reference data and checking that the hashes matched each time.

For timing, we model total acquisition time as:

$$T_{\text{total}} = \sum_{m \in \mathcal{M}} \left(T_m^{\text{acq}} + T_m^{\text{hash}} \right) + T_{\log} \tag{10}$$

where T_m^{acq} is how long module m takes to capture its data, T_m^{hash} is the SHA-256 computation time, and $T_{\log}$ is the overhead from writing and chaining the log entry. Breaking it down this way lets us see exactly which part of a module is the bottleneck as data volumes grow.

C. Results

Table I has the summary. Every module passed on both platforms. Memory dumps came back complete with no missing process entries. Disk images verified bit-for-bit against the source. The network module recorded everything active at capture time — all connections, routes, and caches — without touching a single packet.

TABLE I

MODULE PERFORMANCE SUMMARY

Module	Linux	Windows
Memory Acquisition	Pass	Pass
Disk Imaging	Pass	Pass
Network Forensics	Pass	Pass
SHA-256 Verification	Consistent	Consistent
System Interference	Minimal	Minimal

CPU usage during memory acquisition stayed under 15% on both machines. Disk I/O during imaging stayed within the storage subsystem with no measurable effect on process scheduling. The network module generated no traffic and left no trace in the system's own logs.

D. Comparative Feature Overview

Table II puts the HFS next to Volatility and FTK Imager — two tools that are individually very capable but each covers only part of the acquisition problem.

E. Limitations

Testing has so far covered a narrow slice of the hardware landscape. Network appliances, embedded systems, and Android devices all sit outside what we have validated. Full integration of the three modules under the Master Agent is still being completed — right now each was tested independently rather than as a single automated run. We also need larger and more varied test cases, including machines with encrypted storage and locked-down OS configurations, before we can characterize reliability at any real depth.

TABLE II

FEATURE COMPARISON ACROSS FORENSIC ACQUISITION TOOLS

Feature	Volatility [9]	FTK Imager [3]	HFS (Proposed)
Memory Acq.	Yes	No	Yes
Disk Imaging	No	Yes	Yes
Network Foren-sics	No	No	Yes
Cross-Platform	Ltd	Ltd	Yes
No Install Req.	No	No	Yes
Auto Workflow	No	No	Yes
Hash Verification	No	Yes	Yes
CoC Logging	No	Partial	Yes

Encryption is the most pressing technical gap. The system currently makes no attempt to decrypt BitLocker, VeraCrypt, or LUKS volumes. The memory dump often contains the key material needed to do this — that link between the memory module and the disk module needs to be built. It requires detecting that a volume is encrypted, locating the right key

 DOI : 10.35291/2454-9150.2026.0033

material in the dump, and feeding it into the imaging step before the sector copy begins.

All three modules also assume the investigator has direct physical or local logical access to the target. Remote machines, cloud VMs, and network-attached storage are not currently in scope. Similarly, storage devices with non-standard sector geometries or proprietary firmware — some enterprise NVMe drives, for instance — may not behave correctly under the current write-block and imaging logic.

The network module's snapshot approach is another known limitation. It captures what is active at one moment. Anything that happened before the device was connected — established connections that since dropped, DNS lookups that resolved and were flushed — is gone. We are looking at pulling historical data from OS-level event logs to partially fill that window, but it is not implemented yet.

VI. CONCLUSION

This paper set out to tackle a problem that practitioners deal with on every investigation but rarely see addressed at the tool design level: the fact that forensic acquisition is treated as three separate problems — memory, disk, network — rather than one. The Hardware Forensic Suite treats it as one. Plug in the device, and acquisition runs. The right agent loads for the right OS. Memory goes first because it has to. Disk follows. Network is captured last. Every step is hashed, every hash is chained into the log, and the investigator walks away with a legally defensible evidence package without having manually configured anything.

What makes the integrity model worth highlighting is that it is not bolted on afterward. The SHA-256 verification at each module boundary, the write-block confirmation, the hashchained log — these happen as part of acquisition itself. The chain-of-custody record is not a document that gets filled out after the fact; it is a byproduct of how the system works. That matters in court.

The formal models in this paper — for OS detection confidence, volatility-ordered execution, write-block verification, forensic completeness, network entropy, and log chain validity — give the design a mathematical backbone. They make the system's behaviour predictable, auditable, and benchmarkable. They also make the gap between current state and full production readiness easier to define precisely.

On the practical side, prototype testing showed clean results on both Linux and Windows with interference well within acceptable bounds. The next step is running the full integrated pipeline end-to-end rather than module-by-module, then expanding the test surface to cover more hardware configurations, encrypted volumes, and edge-case OS environments. There is also real work to do on encrypted volume support and remote acquisition. But the foundation is in place, and it holds.

REFERENCES

[1] A. Jaganathan Manonmani, "Data Acquisition Fundamentals and Methodology," *ResearchGate*, 2025. [Online]. Available: https://www. researchgate.net/publication/388492970

[2] M. Abdallah, "A Survey on Data Acquisitions Systems (DAQ)," *International Research Journal of Engineering and Technology (IRJET)*, vol. 6, no. 4, 2009. [Online]. Available: https://www.irjet.net/archives/V6/i4/ IRJET-V6I4289.pdf

[3] K. Ghazinour, "A Study on Digital Forensic Tools," *International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE)*, vol. 5, no. 3, Mar. 2017. [Online]. Available:

https://www.ijarcce.com/upload/2016/march-16/IJARCCE%2026.pdf

[4] S. Raghavan, "Digital Forensic Research: Current State of the Art," *CSI Transactions on ICT*, vol. 1, no. 1, pp. 91–114, 2013. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf &doi= f34f3d09b2deba1f8f876ab5d37f6e2e79f83a2f

[5] H. Fakhouri, "Overview of Challenges Faced by Digital Forensic," *ResearchGate*, 2024. [Online]. Available: https://www.researchgate.net/ publication/368938083

[6] G. Mohay, "Technical Challenges and Directions for Digital Forensics," *Queensland University of Technology ePrints*, 2005. [Online]. Available: https://eprints.qut.edu.au/secure/4254/1/4254.pdf

[7] Z. Shah, "Forensic Investigation of Remnant Data on USB Storage," *Applied Sciences*, vol. 12, no. 16, p. 8042, 2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/16/8042/pdf

[8] B. Cusack, "Including Network Routers in Forensic Investigation," *Edith Cowan University Research Online*, 2014. [Online]. Available: https: //ro.ecu.edu.au/cgi/viewcontent.cgi?article=7682&context=ecuworks

[9] L. Zhang, "Live Memory Acquisition through Firewire," in *Proc. 23rd Chaos Communication Congress*, 2011. [Online]. Available:

https://events.ccc.de/congress/2006/Fahrplan/attachments/1420-paper _live memory acquisition.pdf

[10] A. Korol, "FEAR: A Novel Framework for Representing Digital Forensic," in *Proc. ACM Workshop on Digital Forensics*, 2025. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3339252.3339260

[11] G. Michelet, "Automation for Digital Forensics: Towards a Definition for the Community," *Forensic Science International: Synergy*, vol. 6, p. 100332, 2023. [Online]. Available: https://doi.org/10.1016/j.fsisyn.2023. 100332

[12] H. J. Mohammed, "Automating the Harmonisation of Heterogeneous Data in Digital Forensics," in *Proc. European Conf. on Cyber Warfare and Security (ECCWS)*, 2018. [Online]. Available: https://pureadmin. qub.ac.uk/ws/files/151825025/ECCWS 2018 paper 33.pdf

 DOI : 10.35291/2454-9150.2026.0033

[13] W. Holt, "Development of an Automated Digital Forensics Toolkit for Incident Response," *ResearchGate*, 2021. [Online]. Available: https://www.researchgate.net/publication/331700836

[14] F. Yudha, "A Prototype of Portable Digital Forensics Imaging Tools Using Raspberry Device," *IOP Conference Series: Materials Science and Engineering*, vol. 1098, no. 6, p. 062065, 2021. [Online].

Available: https://iopscience.iop.org/article/10.1088/1757-899X/1098/6/ 062065/pdf

[15] H. J. Mohammed, "Automated Identification of Digital Evidence across Heterogeneous Sources," Ph.D. thesis, University of Plymouth, 2018.

[Online]. Available: https://pearl.plymouth.ac.uk/handle/10026.1/11700

[16] H. Mohammed, "An Automated Approach for Digital Forensic Analysis of Heterogeneous Big Data," *Journal of Digital Forensics, Security and Law*, 2016. [Online]. Available: https://commons.erau.edu/cgi/viewcontent.cgi?article=1000&context=jdfsl

[17] D. Quick and K.-K. R. Choo, "Impacts of Increasing Volume of Digital Forensic Data: A Survey and Future Research Challenges," *Computers & Security*, vol. 45, pp. 34–46, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1742287614000324

[18] R. Montasari, "Next-Generation Digital Forensics: Challenges and Future Paradigms," *ResearchGate*, 2019. [Online]. Available: https://www.researchgate.net/publication/338767419

[19] D.-H. Shin, "Research on Digital Forensics Analyzing Heterogeneous Internet," *Applied Sciences*, vol. 14, no. 5, p. 1751, 2024. [Online]. Available: https://www.mdpi.com/2076-3417/14/5/1751/pdf

[20] P. Reddy, "Usability for Digital Forensics Professionals," in *Proc. Workshop on Security Information Workers (WSIW)*, 2024. [Online]. Available: https://spexlab.org/files/WSIW2024 dfir.pdf

 DOI : 10.35291/2454-9150.2026.0033