DEVELOPMENT TEAM PROJECT REPORT

Module: Intelligent Agents

Group: Group B

Domain: Digital Forensics – File Identification, Integrity Verification, Archiving & Transmission

Table of Contents

1. Executive Summary

The report is a multi-agent system (MAS) that automates key steps of the digital forensics triage pipeline: file identification, filtering, integrity checking, secure archiving, and transfer. The system breaks down the forensic process into specialised agents: Scanner, Filter, Hash Verifier, Archiver, and Transmitter. Python was selected due to its stable cryptography and filesystem libraries.

2. Introduction

Agent-based computing presents an orderly, independent solution to repetitive and rule-based forensic activities. The intelligent agent views the world around it, seeks objectives, and engages other agents in achieving multifaceted tasks (Wooldridge, 2021).

The latest studies in digital forensic intelligence have shown that the systems of multi-agent and automated triage can substantially enhance the efficiency and the accuracy of processing a large amount of data (Quick & Choo, 2018). This project contributes by framing digital forensics as an agent-oriented pipeline, highlighting how autonomy and modular reasoning can reduce investigator bias and improve evidential consistency.

3. Domain Overview: Digital Forensics

Digital forensics aims to identify, preserve, collect and analyse electronic evidence (Casey, 2011).

Key Domain Requirements

Integrity Preservation

Evidence must remain unaltered. Cryptographic hashing (e.g., SHA-256) is required to validate authenticity (NIST, 2006; Mamede et al., 2019, p. 49).

Chain of Custody

All actions performed on evidence must be transparently logged (Karampidis et al., 2018).

File Diversity

Systems must handle multiple file types, hidden files, metadata-rich formats and system artefacts (Carrier, 2005).

Scalability

Forensic datasets can contain hundreds of thousands of files.

Security

Evidence must be protected at rest and during transfer (Bishop, 2005).

These constraints strongly influence how the agents are designed, ensuring that each step of the pipeline remains forensically sound.

4. System Requirements

Software Requirements

- Python 3.10+
- os, pathlib – directory traversal
- hashlib – SHA-256 hashing
- zipfile / tarfile – archiving

- logging – chain-of-custody logging
- json, csv – metadata storage
- requests / paramiko – secure transmission

Hardware Requirements

- ≥ 8GB RAM forensic workstation

Operating System Requirements

- Windows 10+, Ubuntu 20.04+, macOS Ventura+

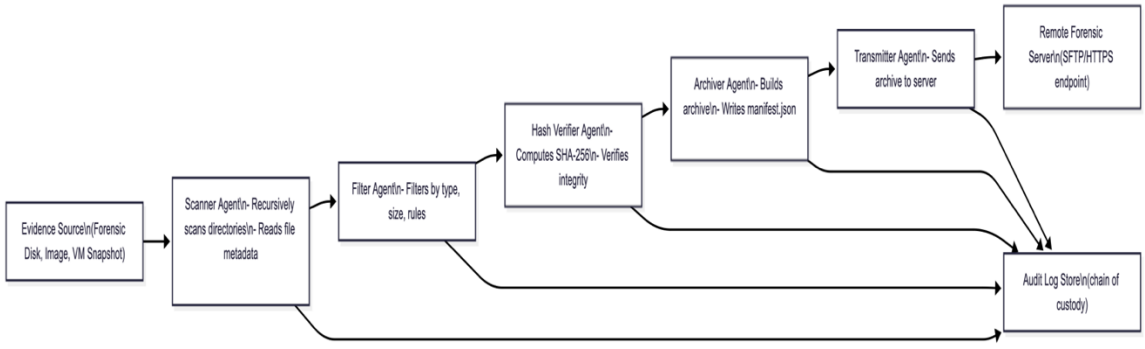
Security Requirements

- HTTPS or SFTP evidence transmission

5. Multi-Agent Architecture

The system adopts a pipeline-based multi-agent architecture in which each agent performs a discrete forensic function and communicates results to the next component.

Figure 1. Multi-Agent Pipeline Architecture: System architecture showing sequential interaction between agents and logging functions.



5.1 Agent Roles

1. Scanner Agent

- Recursively enumerates directories
- Extracts and records file metadata
- Emits file events to the Filter Agent

2. Filter Agent

- Evaluates file types using extensions, MIME types and magic numbers
- Passes accepted files to the Hash Verifier

3. Hash Verifier Agent

- Computes SHA-256 hashes
- Detects corruption (hash mismatches)

4. Archiver Agent

- Creates deterministic ZIP/TAR archives
- Inserts manifest.json for metadata
- Ensures raw evidence is not modified

5. Transmitter Agent

- Sends final archives to remote servers via SFTP/HTTPS

5.2 Agent Reasoning Model (BDI-Inspired)

- **Beliefs:** File paths, metadata, hash states
- **Desires:** Extract and preserve evidence accurately
- **Intentions:** Perform scanning, filtering, hashing and archiving

The agents share a lightweight forensic ontology that defines core concepts such as file, hash, evidence item, and archive. This ensures consistent interpretation of metadata and supports coherent decision-making across agents.

5.3 Architecture Benefits

- High modularity
- Forensic soundness
- Scalable and parallelisable
- Reduced contamination risks
- Easy maintenance and agent replacement

6. Design Decisions & Methodology

Language Selection — Python

Chosen for its mature cryptographic, filesystem and networking libraries, reproducibility and clear syntax (Casey, 2011).

Justification of Library Choices

- **hashlib vs PyCryptodome:** hashlib provides stable, built-in, FIPS-aligned implementations, reducing dependency risks.
- **paramiko vs SCP:** SFTP offers integrity protection and packet-level security, essential for legal-grade evidence transfer.
- **zipfile / tarfile:** Produce deterministic archives, facilitating verification and repeatability.

Hash Algorithm — SHA-256

NIST-recommended, collision-resistant and widely admissible in court (NIST, 2006; Wang & Yu, 2017).

These choices reflect industry-grade forensic practice, aligning the agent architecture with operational requirements used in real investigations.

Methodology Steps

1. Directory enumeration
2. Forensic filtering
3. SHA-256 hashing
4. Metadata extraction
5. Archive creation
6. Secure transfer
7. Chain-of-custody logging

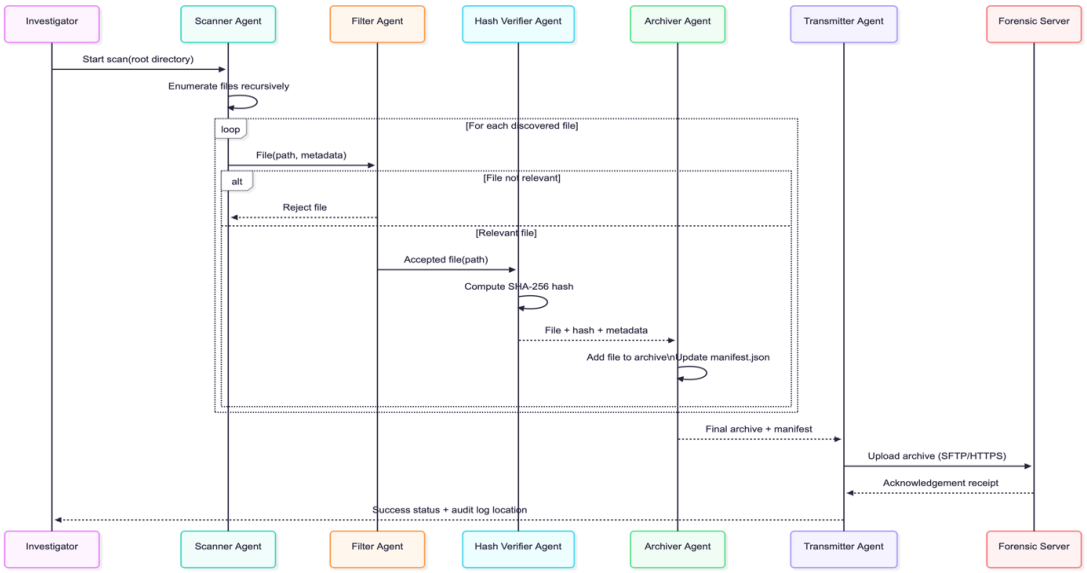
This mirrors standard forensic triage procedures (Palmer, 2001).

7. Workflow of the Forensic Pipeline

1. **Scan** — Identify all files recursively.
2. **Filter** — Apply forensic rules.
3. **Hash** — Compute SHA-256.
4. **Archive** — Bundle files, generate metadata.
5. **Transmit** — Securely send archives.
6. **Audit** — Log all actions for chain-of-custody.

8. Approaches, Patterns & Paradigms

Figure 2. Sequence Diagram of Evidence Processing: Flow of Investigator, Scanner, Filter, Hash Verifier, Archiver and Transmitter agent interactions.



Agent Paradigms

- **Reactive agents** — ideal for real-time filtering (Brooks, 1991).
- **BDI agents** — structured, goal-oriented decision-making (Rao & Georgeff, 1995).

Software Engineering Patterns

- **Factory Pattern** — selects hashing/scanning strategies dynamically.
- **Observer Pattern** — event-driven inter-agent communication.
- **Pipeline Pattern** — structured evidence processing stages.

This alignment ensures that even without full ACL implementation, the agents maintain clarity of intention, mirroring the dialogue patterns demonstrated in Unit 6's KQML/KIF examples.

9. Challenges & Mitigations

Challenge	Impact	Mitigation
Permission restrictions	Missed files	Graceful fallbacks + audit logs
Corrupted files	Integrity errors	Hash mismatch detection
Large datasets	Slow performance	Concurrency + batching
Hidden/system files	Missed evidence	Signature-based detection
Legal compliance	Risk to admissibility	Detailed chain-of-custody

10. **Strengths & Weaknesses**

Strengths

- Modular and maintainable
- Parallelisable
- Forensically sound hashing
- Strong audit logging
- Platform independent

Weaknesses

- Python slower than compiled languages
- No distributed execution
- No tamper-evident timestamping
- Single-host limitation

11. Critical Evaluation

The offered architecture is very consistent with the best practices of forensic integrity preservation, metadata documentation, and chain-of-custody needs (Casey, 2011; Garfinkel, 2013, p. S70). It is modular in nature, thus enhancing its auditability as every agent operation is independently verifiable. A multi-agent architecture introduces coordination overhead, as each agent's communication and event-triggered interactions can increase latency compared to tightly integrated monolithic tools (Wooldridge, 2021). However, unlike KQML/KIF-based communication, the current implementation lacks semantic-level expressiveness, which may limit future integration with ontology-driven forensic systems. This highlights a design trade-off between implementation simplicity and long-term extensibility, a recurring theme in MAS engineering. A future enhancement could incorporate a lightweight ACL layer, enabling richer performatives and ontology-driven interoperability when the system scales or integrates with external forensic platforms.

Recent studies in DFIR also indicate that agent-based and automated triage significantly decreases time-to-evidence, particularly in large-scale datasets (Quick & Choo, 2018).

Future Enhancements

Distributed multi-agent scanning across multiple hosts

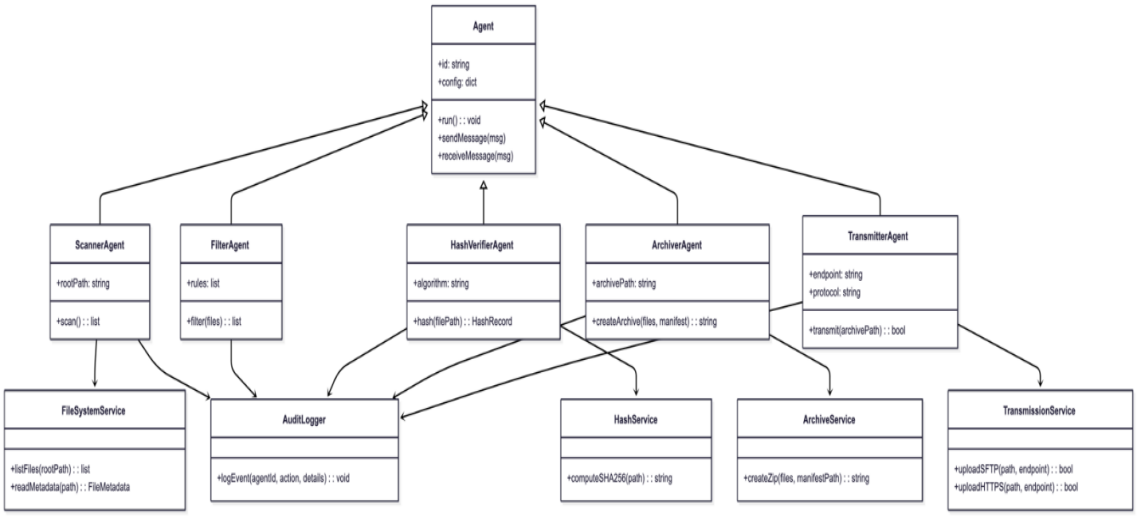
Machine-learning classification agent for prioritising high-value artefacts

Priority-based agent reasoning for adaptive triage

Cryptographic timestamping and blockchain audit trails

Such advances would bring the system to a healthy, scalable forensic automation platform.

Figure 3. UML Component/Class Diagram: Structural relationships between agents, supporting services (FileSystemService, HashService, ArchiveService, TransmissionService), and the AuditLogger. Shows inheritance, message flow, and modular separation of responsibilities within the forensic multi-agent system.



12. Conclusion

The present report introduced a multi-agent system of digital forensic triage, which offered automated scanning, filtering, hashing, archiving, and secure transit. The system is consistent with forensic standards, it supports modularity, and provides a reasonable basis for future distributed and intelligent extensions. It may become a fully operational forensic system of automation with more forensic intelligence agents and tamper-evident logging mechanisms. Integrating structured ACL concepts from Unit 6 would further enhance agent coordination, ensuring semantic clarity in future distributed deployments.

13. References

Altheide, C. & Carvey, H. (2020) Digital Forensics with Open Source Tools. Syngress.

Bellifemine, F., Caire, G. & Greenwood, D. (2007) Developing Multi-Agent Systems with JADE. Wiley.

Bishop, M. (2005) Introduction to Computer Security. Pearson.

Brooks, R.A. (1991) 'Intelligence without representation', Artificial Intelligence, 47(1), pp. 139–159.

Carrier, B. (2005) File System Forensic Analysis. Addison-Wesley.

Casey, E. (2011) Digital Evidence and Computer Crime. 3rd edn. Academic Press.

Gamma, E. et al. (1994) Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

Garfinkel, S. (2010) 'Digital forensics research: The next 10 years', Digital Investigation, 7, pp. S64–S73.

Garfinkel, S. (2013) 'Digital media triage with bulk_extractor', Digital Investigation, 9, pp. S65–S73.

Karampidis, K. et al. (2018) 'Evaluating secure evidence handling', JDFSL, 13(2), pp. 23–34.

Kent, K. et al. (2006) NIST SP 800-86: Forensic Techniques in Incident Response.

Mamede, N. et al. (2019) 'Review of hashing techniques', FSI: Digital Investigation, 28, pp. 45–56.

Palmer, G. (2001) Road Map for Digital Forensic Research. DFRWS.

Quick, D. & Choo, K.-K.R. (2018) 'Digital forensic intelligence', Future Generation Computer Systems, 79, pp. 376–389.

Rao, A.S. & Georgeff, M.P. (1995) 'BDI agents', ICMAS, pp. 312–319.

Russell, S. & Norvig, P. (2021) Artificial Intelligence: A Modern Approach. 4th edn. Pearson.

Wang, X. & Yu, H. (2017) ‘Integrity verification for digital evidence’, ACM TISSEC, 20(4).

Winikoff, M. (2017) ‘Challenges for engineering MAS’, AAMAS, 31(4), pp. 779–817.

Wooldridge, M. (2021) An Introduction to MultiAgent Systems. Wiley.

Wurman, P.R., D’Andrea, R. & Mountz, M. (2008) ‘Coordinating cooperative robots’, AI Magazine, 29(1), pp. 9–19.