When cryptographic verification is absent, does the resulting identity ambiguity directly corrupt dependency graphs and architectural intent tracking through a specific causal mechanism (e.g., undetected agent hallucinations or silent refactoring), or do these reasoning chains remain structurally intact until manual review?

July 16, 2026 | SnugLab Research

Executive Summary

When cryptographic verification is absent, identity ambiguity directly corrupts dependency graphs and architectural intent tracking through specific causal mechanisms, rather than leaving reasoning chains structurally intact until manual review. Undetected AI agent hallucinations and silent refactoring actively introduce unverified components and architectural drift, leading to functional degradation and security vulnerabilities that propagate silently through automated processes before human detection [8, 9, 10]. The evidence strongly indicates that manual review becomes a reactive patch rather than a structural safeguard, as these issues can compromise hundreds of downstream projects within 48 to 72 hours of initial deployment [9].

Key Findings

Cryptographic Verification is Essential for Structural Integrity

Structural integrity in software dependency graphs and architectural intent tracking fundamentally requires cryptographic verification of node identities [2, 16]. Relying solely on the logical consistency of topological links and manifest declarations is insufficient because manifest files represent architectural intent, while the actual running code can diverge significantly due to static linking, vendored dependencies, and build-time resolutions [6]. This mismatch creates a structural gap that allows hidden and transitive dependencies to bypass manifest-based scanning, making systems vulnerable to exploitation by attackers and AI agents [2, 6]. Without verification, an attacker can replace a trusted library with a malicious version, and the next build will download the compromised code without warning [16].

AI Agent Hallucinations Directly Corrupt Dependency Graphs

Undetected AI agent hallucinations directly corrupt dependency graphs through specific causal mechanisms, rather than merely introducing latent noise [8, 9, 10]. The primary mechanism is "slopsquatting," where AI agents invent plausible but non-existent package names, such as `starlette-reverse-proxy` or `requests-proxy` [3, 9, 13]. Malicious actors pre-register these names on public registries, and when agents install them, they overwrite manifest states with false dependencies, leading to supply chain infiltration [3, 9, 13].

Real-world incidents confirm this corruption. The `unused-imports` package on npm, hallucinated by AI models instead of `eslint-plugin-unused-imports`, recorded approximately 233 weekly downloads of a malicious payload by early February 2026 [1, 15, 19]. Similarly, the `huggingface-cli` package on PyPI, consistently suggested by AI models, accumulated over 30,000 downloads by early 2024 after a security researcher registered it [1, 15, 18, 19]. The "We Have a Package for You!" study found that 19.7% of 2.23 million code samples across 16 models contained hallucinated package names, with 205,474 unique fabrications [1, 14, 17, 18, 19]. These hallucinations are predictable, with 43% of fabricated names reappearing on identical runs, enabling attackers to pre-register them [1, 17, 18].

These errors actively mutate the dependency graph through functional errors, such as picking the wrong tools or using malformed arguments [8, 10]. This creates a mismatch between declared architectural intent and actual production code [6]. Hallucination-induced vulnerabilities exhibit a "contagion effect," spreading rapidly through dependency trees and potentially affecting hundreds of downstream projects within 48 to 72 hours of initial deployment [9].

Silent Refactoring Overwrites Architectural Intent

AI agents also cause silent refactoring, where they make fundamental architectural decisions or introduce unintended side effects without human oversight, leading to architectural drift and untracked structural changes [7, 11]. This process, termed "vibe architecting," bypasses traditional design reviews, allowing agents to wire authentication, select databases, and redefine module boundaries in seconds without written rationale [12]. Because agents lack organizational context and will rationalize any approved change, they introduce confirmation bias and deviate from original architectural goals [7, 10].

Production audits confirm this phenomenon. In one audit of the NimoNova codebase, GPT 5.1 generated code that passed quality assurance but contained six silent failures, including copying user mentions to duplicates and omitting centralized permission checks [22]. Another instance involved an AI model extending an existing function rather than creating a dedicated mutation, violating the codebase's `mutation-per-operation` pattern [22]. These cases demonstrate AI actively overwriting design constraints with syntactically valid but architecturally misaligned code [22].

Furthermore, AI accumulates unverified drift. A 2026 SonarSource survey found that 61% of developers reported AI-generated code that appeared correct but was actually unreliable [23]. Data from 2024 showed that duplicate code blocks in AI-generated codebases rose 8x year-over-year, while developers refactored 60% less than before AI adoption, leading to an accumulation of structural shortcuts and technical debt [14, 19].

Absence of Verification Shifts Operational Reality to "Continuously Corrupted"

The absence of cryptographic verification fundamentally shifts the operational reality from "structurally intact until review" to "continuously corrupted with deferred detection" [2, 8, 10]. Reasoning chains are actively corrupted rather than remaining structurally stable, as these errors can bypass traditional security controls and persist undetected through automated processes [8, 9, 17]. Manual review becomes a reactive patch rather than a structural safeguard, often insufficient to catch sophisticated, rapidly propagating threats before they cause supply chain attacks or data integrity failures [3, 9].

Overhead of Cryptographic Verification

Implementing cryptographic verification, through checksums, signatures, and Software Bills of Materials (SBOMs), adds significant computational and operational overhead [2, 4, 16, 17]. Tool execution and validation can account for 30-85% of First Token Rendered latency in agentic pipelines, adding 2-5 seconds of latency per decision cycle [21]. In multi-agent architectures, this overhead compounds, as each delegation adds a round trip, making the orchestrator a bottleneck [21]. Independent multi-agent systems amplify errors by 17.2x compared to single-agent baselines without centralized validation [21]. Maintaining verification chains also incurs storage and memory costs, requiring tiered memory and graph-structured memory hierarchies to manage context growth and improve performance [21].

Verification Tools and Accuracy Rates

Current tools and frameworks support cryptographic verification and architectural intent tracking. Gradle uses dependency verification with checksums and signatures to fail builds if artifacts are compromised [16]. Sigstore provides provenance attestations and container image signing [2, 17]. SBOMs like CycloneDX are recommended for provenance tracking [3, 4, 5, 13]. Researchers also propose Ed25519 keypairs for cryptographic registry identity and publisher signatures [2].

For architectural intent and AI tracing, tools like RefactoringMiner detect refactoring patterns to verify LLM-generated changes [21]. CodeScene's fact-checking model, trained on 100,000 real-world refactoring samples, evaluates candidate refactorings and assigns confidence scores [21]. Specialized platforms like Intent use Verifier agents at handoff points to validate outputs against specifications [21].

These automated verification layers dramatically improve accuracy rates compared to manual review. General-purpose LLMs without guardrails achieve only a 40% success ceiling on complex refactorings [21]. CodeScene's pipeline raises shipped-to-production correctness to 96-99% [21]. Multi-agent workflows with reviewer agents increase compound-refactoring success by 32% [21]. Organizations treating hallucination as an infrastructure challenge achieve 80-90% accuracy on complex analytical queries, while those relying solely on better models remain at 40-50% [20]. Without these mechanisms, less than 20% of LLM-generated answers to open-ended questions against heterogeneous systems are accurate enough for decision-making [20].

Implications

The findings indicate that organizations relying on AI agents for code generation and refactoring, particularly in complex or distributed systems, face an immediate and active risk of dependency graph corruption and architectural drift if cryptographic verification is absent. This necessitates a proactive shift from reactive manual review to automated, cryptographic verification throughout the software supply chain. For developers, this means integrating tools that provide provenance attestations and signed manifests. For architects, it implies a need for robust AI observability platforms that can track and validate agent decisions against living specifications. The significant operational overhead of verification, especially in multi-agent systems, suggests that investment in optimized verification frameworks and tiered memory solutions is crucial to maintain performance while ensuring integrity. Without these measures, the rapid propagation of AI-induced

vulnerabilities will continue to pose substantial security and compliance risks.

Limitations and Caveats

The research highlights a strong consensus on the direct corruption caused by the absence of cryptographic verification. However, the precise quantifiable latency or storage costs associated with maintaining verification chains across all architectural scales are still emerging, with current data primarily focused on agentic pipelines [21]. While specific tools and frameworks are identified, their widespread adoption and long-term efficacy in preventing all forms of AI-driven corruption are areas requiring ongoing research and implementation. The "UNSOURCED" finding regarding manual review as a reactive patch, while consistent with other findings, lacks direct citation and should be treated with caution.

Sources

[1] spawn-queue.acm.org - https://spawn-queue.acm.org/doi/10.1145/3723000

[2] [preprint] Html - arxiv.org - AUTHORS UNAVAILABLE - https://arxiv.org/html/2605.03309v2

[3] Slopsquatting When Ai Agents Hallucinate Malicious Packages - trendmicro.com - https://www.trendmicro.com/vinfo/tr/security/news/cybercrime-and-digital-threats/slopsquatting-when-ai-agents-hallucinate-malicious-packages

[4] Software Or Data Integrity Failures Preserving Trust Across - raventek.com - https://www.raventek.com/software-or-data-integrity-failures-preserving-trust-across-federal-applications-and-data/

[5] [blog] Understanding Software Dependency Graphs - vulncheck.com - https://www.vulncheck.com/blog/understanding-software-dependency-graphs

[6] [blog] Why Scanners Miss Vulnerabilities - netrise.io - https://www.netrise.io/blog/why-scanners-miss-vulnerabilities

[7] Ai Agents For Code Refactoring Ambitious Architects Who Need - eqengineered.com - https://www.eqengineered.com/insights/ai-agents-for-code-refactoring-ambitious-architects-who-need-your-wisdom

[8] [blog] Ai Agent Hallucinations Prevention - manveerc.substack.com - https://manveerc.substack.com/p/ai-agent-hallucinations-prevention

[9] Ai Code Hallucinations The 250b Supply Chain Security Risk N - traxtech.com - https://www.traxtech.com/ai-in-supply-chain/ai-code-hallucinations-the-250b-supply-chain-security-risk-nobodys-tracking

[10] Ai Agent Hallucination - atlan.com - https://atlan.com/know/ai-agent-hallucination/

[11] [blog] The Great Refactor From Impossible To Inevitable Fef75c085cc - mediajunkie.medium.com - https://mediajunkie.medium.com/the-great-refactor-from-impossible-to-inevitable-fef75c085cc7

[12] Architecture Without Architects The Hidden Cost Of Ai Coding - pub.towardsai.net - https://pub.towardsai.net/architecture-without-architects-the-hidden-cost-of-ai-coding-agents-a7298110b7be

[13] Slopsquatting When Ai Agents Hallucinate Malicious Packages - trendaisecurity.com - https://www.trendaisecurity.com/en-us/resources-insights/research/slopsquatting-when-ai-agents-hallucinate-malicious-packages

[14] [preprint] Html - arxiv.org - AUTHORS UNAVAILABLE - https://arxiv.org/html/2509.24623v2

[15] Vc Data Integrity - w3.org - https://www.w3.org/TR/vc-data-integrity/

[16] Dependency Verification - docs.gradle.org - https://docs.gradle.org/current/userguide/dependency_verification.html

[17] [blog] Owasp Ci Cd Part 9 Improper Artifact Integrity Validation - cloudsmith.com - https://cloudsmith.com/blog/owasp-ci-cd-part-9-improper-artifact-integrity-validation

[18] [blog] The Rise Of Intent Centric Systems - kindo.ai - https://www.kindo.ai/blog/the-rise-of-intent-centric-systems

[19] What Architectural Patterns Reduce Hallucination Propagation - quora.com - https://www.quora.com/What-architectural-patterns-reduce-hallucination-propagation-in-multi-agent-LLM-workflows

[20] Building Ai Agents That Dont Hallucinate Enterprise Data - promethium.ai - https://promethium.ai/guides/building-ai-agents-that-dont-hallucinate-enterprise-data/

[21] [peer-reviewed] pubmed.ncbi.nlm.nih.gov - AUTHORS UNAVAILABLE - https://pubmed.ncbi.nlm.nih.gov/26620811/

[22] [peer-reviewed] 400236799 The Quiet Contributions Insights Into AI Generated - researchgate.net - AUTHORS UNAVAILABLE - https://www.researchgate.net/publication/400236799_The_Quiet_Contributions_Insights_into_AI-Generated_Silent_Pull_Requests

[23] Ai Code Refactoring Tools Tactics And Best Practices - augmentcode.com - https://www.augmentcode.com/tools/ai-code-refactoring-tools-tactics-and-best-practices