

CRYPTOMAP: Evidence-Driven Cryptographic Dependency Mapping for Post-Quantum Migration

Komal Vasant Rokade¹ · Rohit S Raut¹ · Saniya Adinath Patil¹ · Sneha Divakar Karande¹ ·
Shreyash Uday Mhetar¹ · Shweta Battas Gidde¹

¹Department of Computer Science and Engineering (Data Science),

Dr. D. Y. Patil Pratishthan's College of Engineering, Salokhe Nagar, Kolhapur, Maharashtra, India

Komal Vasant Rokade: komalrokade17@gmail.com · Rohit S Raut: rohitraut8680@gmail.com · Saniya Adinath Patil: patilsaniya357@gmail.com ·
Sneha Divakar Karande: snehakarande32@gmail.com · Shreyash Uday Mhetar: shreyashmhetar@gmail.com · Shweta Battas Gidde:
shwetagidde7@gmail.com

Abstract—The transition from conventional public-key cryptography to post-quantum cryptography (PQC) is becoming an important requirement for organizations operating large and long-lived software systems. Enterprise cryptographic mechanisms are distributed across source code, libraries, APIs, certificates, keys, TLS/SSL configurations, and infrastructure, making algorithm identification alone insufficient to understand migration impact. This paper presents CRYPTOMAP, an evidence-driven framework that combines repository-level cryptographic discovery, dependency-graph construction, deterministic migration-impact analysis, and bounded local artificial intelligence for explanation. A progressive repository-reduction pipeline filters repository content before evidence is normalized and represented as a repository-scoped graph. Blast radius, impact, and migration ordering are computed deterministically, while a planned local language model is intended to explain the resulting verified context rather than determine graph facts. The implementation uses Neo4j, PostgreSQL, FastAPI, and n8n. The current implementation was validated with 151 automated tests, all passing. These results establish implementation-level correctness of the tested software contracts, not universal cryptographic-detection completeness or enterprise migration effectiveness. The planned local-AI explanation and grounding workflow remains future work.

Index Terms—post-quantum cryptography, PQC migration, dependency graph, cryptographic discovery, static analysis, blast radius, migration ordering, provenance, bounded context, large language models

I. INTRODUCTION

The transition from conventional public-key cryptography to post-quantum cryptography (PQC) is becoming an important requirement for organizations that operate large and long-lived software systems. The migration is not simply a matter of replacing one cryptographic algorithm with another. Cryptographic mechanisms are often spread across application source code, third-party libraries, APIs, certificates, keys, TLS/SSL configurations, truststores, keystores, hardware security modules (HSMs), and infrastructure components. As a result, changing one cryptographic primitive can affect multiple connected software and infrastructure assets.

A major challenge in enterprise PQC migration is therefore identifying both the scope and the order of the required changes. An organization may be able to locate RSA, elliptic-curve cryptography, certificates, or cryptographic libraries in

a repository, but simply locating these artifacts does not show which other assets depend on them. A change to a key, certificate, API, or cryptographic service can propagate through several dependency levels and affect downstream applications or services. Treating these assets separately can therefore provide an incomplete view of migration impact.

Another challenge comes from the increasing use of large language models (LLMs) for software analysis. LLMs can summarize source code and generate useful explanations, but allowing an LLM to independently determine dependency relationships, impact, or migration order can introduce unsupported conclusions. This is particularly important in security-sensitive migration analysis, where a plausible explanation may not necessarily be supported by repository evidence. Sending an entire enterprise repository to an external model may also raise concerns related to source-code confidentiality, context limits, cost, and reproducibility.

This work presents **CRYPTOMAP**, an evidence-driven framework for post-quantum cryptography migration analysis based on dependency graphs and bounded local artificial intelligence. The central design principle is to separate authoritative analysis from language-model interpretation. Deterministic software analysis is used to discover cryptographic evidence, normalize assets, construct dependency relationships, calculate blast radius and impact, and determine migration ordering. The resulting verified information is then reduced into a bounded context that can be provided to a local language model for explanation. In this design, the language model acts as an interpretation layer rather than as the source of truth for graph analysis.

CRYPTOMAP follows a progressive repository-reduction pipeline. Instead of providing an entire repository directly to a language model, the system first inventories and filters repository content, identifies cryptographic evidence, normalizes relevant assets, constructs a dependency graph, performs deterministic analysis, and extracts only the subgraph and evidence relevant to the requested analysis.

The current implementation provides a deterministic migration-analysis core consisting of repository scanning, cryptographic evidence discovery, evidence normalization, graph construction, confidence and provenance tracking, blast-

radius analysis, impact analysis, migration ordering, cycle handling, bounded verified-context generation, PostgreSQL-backed job management, FastAPI services, and n8n-based orchestration.

The primary contributions are:

- 1) An evidence-driven PQC dependency-graph architecture that represents cryptographic and infrastructure assets together with their dependency relationships and supporting evidence.
- 2) A deterministic migration-impact analysis approach for dependency-based blast-radius analysis, impact assessment, and migration ordering without delegating authoritative decisions to an LLM.
- 3) A progressive repository-reduction and bounded-context mechanism that limits downstream reasoning to migration-relevant information.
- 4) A confidence- and provenance-aware analysis model that maintains traceability from source-level observations to graph assets and analysis results.
- 5) A separation between deterministic security analysis and local-AI explanation, allowing future language-model components to operate on verified analysis results rather than independently determining graph facts.

II. RELATED WORK

A. Enterprise PQC Migration and Cryptographic Agility

PQC migration in enterprise environments involves more than replacing individual cryptographic algorithms. Cryptographic functionality can be distributed across applications, libraries, certificates, keys, protocols, infrastructure components, and organizational systems. Hasan et al. [1] propose a framework for migration to post-quantum cryptography that incorporates security dependency analysis and graph-theoretic techniques. Baseri et al. [2] present an evaluation framework for quantum-security risk assessment during transition toward quantum-safe systems. Hirsch and Raab [3] investigate architecture-derived Cryptographic Bill of Materials (CBOM) approaches, while Adebesin [4] examines cryptographic agility in enterprise software systems.

Collectively, these studies indicate that effective PQC migration requires visibility into cryptographic assets and the relationships between those assets and dependent systems. However, the reviewed approaches do not by themselves define the complete repository-to-analysis workflow implemented by **CRYPTOMAP**.

B. Cryptographic Discovery and Static Analysis

Automated program analysis has been investigated for identifying cryptographic API usage and detecting implementation problems. Kru"ger et al. [5] introduced CogniCrypt, a specification-driven approach for supporting developers in using cryptographic APIs. CryptoGuard provides large-scale cryptographic program analysis and vulnerability detection [6]. Schlichtig et al. [7] introduced CamBench for evaluating cryptographic API misuse-detection tools.

More recent research has explored LLM-assisted cryptographic discovery. Hirsch et al. [8] investigate collaborative LLM-based detection of cryptographically relevant software packages. In contrast, **CRYPTOMAP** assigns authoritative discovery and graph-analysis responsibilities to deterministic processing and restricts the local language model to interpretation of verified analysis results.

C. Graph-Based Dependency and Impact Analysis

Software systems naturally form dependency networks in which components can depend directly or indirectly on other components. Fritz et al. [9] model software dependencies as directed networks and investigate systemic risk propagation. Zhang et al. [10] examine hidden dependencies in Java software. Sharma et al. [11] study relationships between design and architecture smells, while Pashchenko et al. [12] demonstrate the importance of distinguishing vulnerable dependencies from dependencies that are actually reached or used.

These studies motivate dependency-aware analysis in which the presence of an artifact is not considered sufficient evidence of operational impact. **CRYPTOMAP** applies dependency traversal to a cryptographic asset graph so that potential downstream effects of changing a cryptographic component can be analyzed deterministically.

D. LLMs for Code Analysis and Grounded Reasoning

LLMs have increasingly been applied to software engineering and code-analysis tasks, but their ability to reliably reason about program structure remains an important research concern. Fang et al. [13] evaluate LLMs for code analysis and report limitations in software-security analysis. Wang et al. [14] introduce SGCR, a specification-grounded framework for trustworthy LLM code review. Abtahi and Azim [15] investigate augmentation of LLM-based software engineering with static code analysis. Nguyen and Nguyen [16] review the use of LLMs in software engineering.

Based on these findings, **CRYPTOMAP** adopts a deliberately constrained role for its local language model. Deterministic analysis remains responsible for dependency traversal, blast-radius computation, impact assessment, and migration ordering. The language model receives a bounded, verified context containing those results and is intended to explain them rather than independently construct authoritative graph relationships.

E. Official Guidance and Cryptographic Discovery Standards

The NIST/NCCoE Migration to Post-Quantum Cryptography project identifies cryptographic discovery and inventory as important capabilities for understanding where and how cryptography is used [17]. Joint CISA, NSA, and NIST guidance similarly emphasizes identification of quantum-vulnerable algorithms and their use across applications, libraries, protocols, and dependencies [18]. The Cloud Security Alliance provides additional context for PQC key-management structures and migration readiness [19]. NIST SP 1800-38B addresses

public-key application discovery tools and provides an official discovery-oriented baseline [20].

F. Research Gap

The reviewed literature covers several important parts of the PQC migration problem, including cryptographic discovery, dependency analysis, migration risk assessment, static analysis, and grounded LLM-based reasoning. However, these capabilities are generally considered separately. A practical migration workflow still needs to connect repository-level evidence to normalized cryptographic assets, dependency relationships, deterministic impact analysis, and an explanation mechanism that does not change the underlying analysis results.

CRYPTOMAP addresses this integration by combining these stages in a single evidence-driven pipeline. Repository content is first reduced to migration-relevant information, after which cryptographic evidence is normalized into repository-scoped assets and relationships. Deterministic graph algorithms then calculate dependency traversal, blast radius, impact, and migration ordering. Only the resulting verified subgraph and supporting evidence are intended to reach the local language-model layer.

This separation is important because the language model is not required to reconstruct the dependency graph or make authoritative migration decisions. Instead, it operates on information that has already been produced and validated by deterministic analysis.

G. Source-to-Contribution Comparison

The reviewed studies address different parts of cryptographic discovery, dependency analysis, migration assessment, and grounded AI reasoning. To make the positioning of **CRYPTOMAP** explicit, Table I provides a descriptive comparison across six consolidated capabilities. The comparison is intended to show differences in scope rather than to rank the prior studies.

III. PROBLEM FORMULATION AND RESEARCH QUESTIONS

A. Problem Formulation

Let a repository be represented as

$$R = \{f_1, f_2, \dots, f_n\}, \quad (1)$$

where each f_i represents a file or repository artifact. A deterministic discovery process extracts cryptographic evidence:

$$E = \{e_1, e_2, \dots, e_m\}. \quad (2)$$

Each evidence item may contain its source file, location, detected cryptographic algorithm or API, symbol or configuration element, evidence type, and repository identifier.

The evidence is subsequently normalized into graph assets:

$$V = \{v_1, v_2, \dots, v_k\}, \quad (3)$$

and dependency relationships:

$$E_G \subseteq V \times V, \quad (4)$$

forming a repository-scoped dependency graph

$$G = (V, E_G). \quad (5)$$

For a selected target asset v_t , the downstream dependency set is

$$D(v_t, d) = \{v \in V \mid v_t \rightsquigarrow v, \text{depth}(v_t, v) \leq d\}, \quad (6)$$

where d is the configured traversal depth. This set forms the basis for blast-radius analysis.

When dependency cycles are present, strongly connected components (SCCs) are identified and condensed into a directed acyclic graph (DAG) before migration ordering is derived.

The resulting analysis is represented as a verified context containing the target asset, affected assets, dependency relationships, impact information, migration-order information, supporting evidence, confidence, provenance, and warnings. A local language model receives only this bounded context and is not responsible for establishing graph relationships, computing blast radius, determining impact scores, or selecting migration order.

B. Research Objectives

The study has five primary objectives:

- 1) automated cryptographic evidence discovery;
- 2) evidence-driven dependency modeling;
- 3) deterministic migration-impact analysis;
- 4) progressive repository reduction and bounded-context generation; and
- 5) evidence-grounded local-AI explanation.

Each primary objective is mapped to one research question so that the research scope remains explicit and traceable.

C. Research Questions

RQ1 – Cryptographic Discovery: Can cryptographic evidence from heterogeneous software repositories be systematically extracted and normalized into migration-relevant assets?

RQ2 – Evidence-Driven Dependency Modeling: Can heterogeneous cryptographic evidence be normalized into repository-scoped assets and dependency relationships in a deterministic and traceable manner?

RQ3 – Deterministic Migration-Impact Analysis: Can deterministic graph analysis identify the downstream dependency scope, impact information, and migration ordering associated with a selected cryptographic asset?

RQ4 – Progressive Reduction and Verified Context: Can progressive repository reduction produce a bounded context that retains the evidence required to explain migration impact?

RQ5 – Evidence-Grounded Local-AI Explanation: How can a local language model be constrained to explain deterministic, verified migration results without independently determining authoritative graph facts?

RQ5 is addressed as an architectural design objective in the current work. The complete Ollama-based explanation and

TABLE I
DESCRIPTIVE COMPARISON OF PRIOR WORK WITH CRYPTOMAP

Source	Crypto.	Evidence/Graph	Migration/Risk	Deterministic	LLM/Grounded	PQC/Enterprise
Hasan et al. [1]	✓	✓	✓	✓	–	✓
Baseri et al. [2]	Partial	Partial	✓	✓	–	✓
Hirsch and Raab [3]	✓	✓	✓	✓	–	✓
Adebesin [4]	Partial	✓	✓	✓	–	✓
Kru'ger et al. [5]	✓	✓	–	✓	–	–
Rahaman et al. [6]	✓	✓	–	✓	–	–
Schlichtig et al. [7]	✓	✓	–	✓	–	–
Hirsch et al. [8]	✓	✓	Partial	Partial	✓	✓
Fritz et al. [9]	–	✓	✓	✓	–	–
Zhang et al. [10]	–	✓	Partial	✓	–	–
Sharma et al. [11]	–	✓	–	✓	–	–
Pashchenko et al. [12]	–	✓	✓	✓	–	–
Fang et al. [13]	–	Partial	Partial	Partial	✓	–
Wang et al. [14]	–	✓	–	✓	✓	–
Abtahi and Azim [15]	–	✓	–	✓	✓	–
Nguyen and Nguyen [16]	–	Partial	–	Partial	✓	–
NIST/NCCoE [17]	✓	✓	✓	✓	–	✓
CISA/NSA/NIST [18]	✓	✓	✓	✓	–	✓
CSA [19]	✓	✓	✓	✓	–	✓
NIST SP 1800-38B [20]	✓	✓	✓	✓	–	✓
CRYPTOMAP	✓	✓	✓	✓	✓	✓

✓ indicates that the capability is explicitly addressed; “Partial” indicates partial coverage; “–” indicates that the capability is not a primary focus of the referenced work. The comparison is descriptive and does not imply that one study is generally superior to another.

TABLE II
ALIGNMENT BETWEEN RESEARCH OBJECTIVES AND RESEARCH QUESTIONS

Research Objective	RQ
Cryptographic evidence discovery	RQ1
Evidence-driven dependency modeling	RQ2
Deterministic migration-impact analysis	RQ3
Progressive reduction and bounded context	RQ4
Evidence-grounded local-AI explanation	RQ5

grounding-validation workflow has not yet been experimentally evaluated, so no effectiveness claim is made for local-AI explanation in the current results.

D. Scope and Assumptions

The study focuses on repository-level PQC migration analysis rather than complete enterprise cryptographic governance. The prototype assumes that relevant cryptographic usage leaves detectable evidence in source code, configuration, dependencies, or supported artifacts. Static discovery is not assumed to be complete for dynamically generated code, undocumented runtime behavior, unavailable binaries, or cryptographic operations outside the analyzed repository.

The graph is repository-scoped so that assets and relationships from different repositories are not unintentionally combined. Evidence provenance is retained to support traceability from graph entities back to source observations. The language model is explicitly outside the authoritative analysis path.

E. Evaluation Strategy

RQ1 is evaluated using controlled demonstration repositories containing known cryptographic artifacts and configurations. RQ2 is evaluated through tests covering asset identity, repository scoping, relationship construction, and provenance

preservation. RQ3 is evaluated through deterministic blast-radius, impact, migration-ordering, and cycle-handling tests. RQ4 is evaluated by measuring bounded-context construction, truncation behavior, and preservation of supporting evidence. RQ5 is addressed through the proposed architectural separation between deterministic analysis and local-AI explanation; complete LLM execution and grounding validation remain future experimental work.

IV. CRYPTOMAP SYSTEM ARCHITECTURE

A. System Overview

CRYPTOMAP is an evidence-driven framework for analyzing the impact of PQC migration within software repositories. The architecture combines deterministic repository analysis, dependency graph construction, migration-impact analysis, and bounded local artificial intelligence for interpretation.

The system progressively reduces a software repository to cryptographically relevant information. The processing pipeline consists of repository scanning, relevant-file identification, cryptographic evidence discovery, evidence normalization, dependency graph construction, deterministic migration analysis, and verified-context generation. The dependency graph is stored in Neo4j, while PostgreSQL is used for repository metadata, job metadata, and execution state. FastAPI provides the application layer and n8n orchestrates execution.

B. Progressive Repository Reduction

Enterprise repositories may contain source code, configuration files, documentation, generated files, binaries, images, build artifacts, dependency metadata, and unrelated content. **CRYPTOMAP** therefore applies progressive repository reduction before generating the final analysis context.

The first stage creates a file inventory. Files are filtered according to cryptographic relevance. Generated files, binary

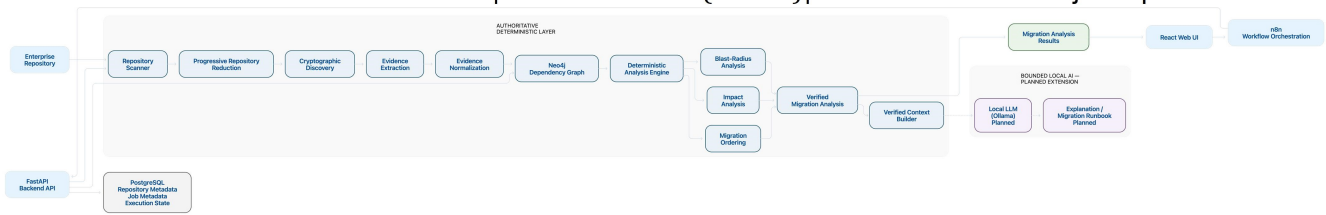


Fig. 1. Overall architecture of CRYPTOMAP, showing progressive repository reduction, cryptographic evidence extraction, repository-scoped dependency graph construction, deterministic migration-impact analysis, and the planned bounded local-AI explanation layer.

artifacts, images, build outputs, and unrelated content can be excluded, while YAML, JSON, properties, XML, environment configuration, dependency manifests, and source-code files can be retained.

The remaining files are grouped into logical units for deterministic cryptographic discovery. The scanner searches for cryptographic libraries, algorithms, API usage, TLS/SSL configuration, certificate and key references, keystore and truststore references, and other migration-relevant indicators. The resulting evidence is normalized into assets and relationships. Only the relevant graph substructure is subsequently used to construct a bounded verified context.

C. Cryptographic Discovery and Evidence Extraction

The discovery component identifies cryptographically relevant information using deterministic scanning. Source-level discovery includes cryptographic library imports, API calls, and algorithm references. Configuration-level discovery includes TLS/SSL settings, certificate references, key references, keystores, truststores, and related security configuration.

Dependency analysis considers cryptographic packages and dependency declarations, while infrastructure evidence may include HSM and key-management references. Examples of detectable technologies include RSA, ECC, ECDSA, AES, SHA, TLS/SSL, X.509 certificates, cryptographic libraries, and key-management references. The objective is to produce structured evidence from available repository content rather than to claim complete detection of every cryptographic dependency in arbitrary enterprise systems.

An evidence record may contain repository identifier, file location, line or source location, symbol or API, identified algorithm or technology, evidence type, and source information. Discovery and graph construction are deterministic components; the LLM is not used to invent cryptographic findings or establish authoritative graph relationships.

D. Evidence Normalization and Graph Construction

Raw evidence can contain duplicate observations, different representations of the same asset, or references requiring identity resolution. Normalization therefore includes deduplication, identity resolution, asset normalization, repository scoping, confidence assignment, and provenance preservation.

After normalization, assets and relationships are represented in Neo4j. The graph contains migration-relevant entities such as applications, certificates, keys, servers, APIs, and HSMs.

Relationship metadata can include confidence and provenance information. This representation enables deterministic traversal while maintaining a traceable connection to repository observations.

E. Dependency Graph Model

The dependency graph is a directed, repository-scoped graph:

$$G = (V, E_G). \quad (7)$$

The principal asset types are Application, Certificate, Key, Server, API, and HSM. Asset metadata can include identifiers, names, criticality, algorithm information, migration status, and repository scope.

A representative dependency chain is

$$\text{HSM} \rightarrow \text{Key} \rightarrow \text{Certificate} \rightarrow \text{Application}. \quad (8)$$

Applications may also depend on other applications or call APIs. Each graph is scoped using a repository identifier, providing an isolation boundary. Relationship confidence and provenance provide additional information about how graph edges were established.

F. Deterministic Analysis Layer

The deterministic analysis layer is the authoritative analytical component. It consists of blast-radius analysis, impact analysis, and migration ordering.

Blast-Radius Analysis: For a selected target asset and traversal depth D , the blast radius is

$$BR(v_t, D) = \{v_i \mid v_i \rightsquigarrow v_t, d_i \leq D\}. \quad (9)$$

Here, d_i is the shortest dependency depth. Multiple paths to the same asset are consolidated by retaining the minimum depth and maximum calculated path-confidence value.

Impact Analysis: The impact analysis calculates a deterministic impact score using asset criticality, evidence confidence, asset type, and dependency depth. The score is reproducible for the same graph, target asset, and configuration. It is an implemented prioritization metric rather than a universal enterprise-risk measure.

Migration Ordering: Migration ordering uses dependency relationships to establish precedence. When cycles exist, strongly connected components are condensed into a directed acyclic graph, after which topological ordering is applied:

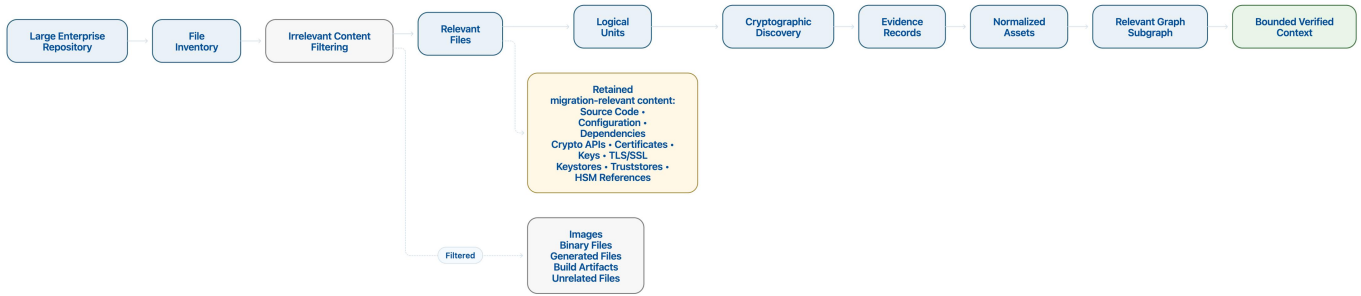


Fig. 2. Progressive repository-reduction pipeline used by CRYPTOMAP to reduce repository-scale input while preserving migration-relevant cryptographic evidence.



Fig. 3. Cryptographic evidence discovery, normalization, and graph construction pipeline used by CRYPTOMAP.

$$G \rightarrow SCC(G) \rightarrow DAG(G) \rightarrow \text{Topological Order.} \quad (10)$$

This avoids arbitrary ordering of individual nodes inside a cycle.

G. Verified Context and Bounded Local AI

The verified-context layer prevents the language model from receiving unrestricted repository information or being treated as the source of authoritative graph facts. The context may contain the selected target asset, affected assets, dependency paths, impact information, migration order, confidence, warnings, supporting evidence, and provenance.

Explicit bounds include relevant-subgraph selection, maximum context size, evidence-item limits, repository scope, and provenance preservation. The resulting context is a compact representation of verified analysis rather than a raw repository dump.

The intended local-AI architecture uses Ollama to run a local LLM. The model is intended to interpret verified context and produce human-readable explanations or migration guidance. It should not independently determine graph relationships, authoritative blast radius, authoritative impact score, or authoritative migration order. The complete Ollama-based explanation and grounding-validation workflow remains a planned extension.

H. Workflow Orchestration and Job Management

n8n is used as the workflow orchestration layer. It coordinates execution between the user-facing interface and FastAPI

backend rather than performing cryptographic discovery or graph analysis.

A typical scan workflow begins with a webhook request, validates the request, registers the repository, creates a scan job, polls the backend for progress, retrieves completed results, or handles failure and timeout conditions. The job lifecycle includes:

QUEUED → SCANNING → DISCOVERING → BUILDING GR

PostgreSQL stores repository metadata, job metadata, and execution state. FastAPI provides application services and Neo4j stores the repository-scoped dependency graph.

I. Security and Repository Isolation

The dependency graph is repository-scoped. Each repository has an identifier and assets and relationships remain within that scope. The provenance chain is:

Source File → Evidence → Graph Asset → Analysis Result → Verifie

Bounded context reduces unnecessary exposure of unrelated repository content. Local inference through Ollama is intended to support processing without sending repository content to an external AI service, but this property remains part of the planned AI extension until the complete workflow is implemented and experimentally evaluated.



Fig. 4. Repository-scoped dependency graph model containing migration-relevant assets and directed dependency relationships.



Fig. 5. n8n orchestration workflow coordinating asynchronous repository analysis, job polling, status handling, and result retrieval.

V. DETERMINISTIC MIGRATION ANALYSIS

A. Blast-Radius Analysis

For a selected cryptographic asset v_t , **CRYPTOMAP** traverses the directed dependency graph up to a configurable maximum depth D , with a default value of 15:

$$BR(v_t, D) = \{v_i \mid v_t \rightsquigarrow v_i, d_i \leq D\}. \quad (11)$$

TABLE III
IMPLEMENTED ASSET-LABEL WEIGHTS

Asset Type	Weight
Application	1.25
API	1.15
Certificate	1.10
Server	1.05
Key	1.00
HSM	1.00

For each reachable asset, the system records shortest graph distance and path confidence. Multiple paths to the same asset are consolidated using minimum depth and maximum calculated path-confidence.

B. Impact Score Calculation

The deterministic impact score is defined as

$$ImpactScore = 10C_t + \sum_{i \in A} \frac{C_i \cdot Conf_i \cdot W_i}{1 + 0.5d_i}. \quad (12)$$

Here, C_t is target criticality, A is the affected-asset set, C_i is affected-asset criticality, $Conf_i$ is path confidence, W_i is the asset-label weight, and d_i is shortest dependency depth.

The implemented weights are shown in Table III. Asset criticality ranges from 1 to 5 and confidence ranges from 0 to 1. The depth attenuation term reduces the contribution of increasingly distant dependencies. The final score is rounded to two decimal places. This is an implemented deterministic prioritization metric, not a universally calibrated risk measure.

C. Dependency-Aware Migration Ordering

The implementation constructs a directed migration graph in which an edge $u \rightarrow v$ represents that u should be migrated before or alongside v . The implemented domain precedence is:

$$HSM \rightarrow Key \rightarrow Certificate \rightarrow Application \rightarrow API \rightarrow Server. \quad (13)$$

The resulting graph is processed using topological traversal. Within the same generation, assets are deterministically ordered using asset-label precedence, descending criticality, and ascending asset identifier.

D. Cycle Handling Using SCC and DAG

Real dependency graphs may contain cyclic relationships. **CRYPTOMAP** identifies strongly connected components and condenses each SCC into a single component, producing a DAG. Topological generations are then generated from the condensed graph. Cyclic groups can therefore be treated as migration units while preserving dependency ordering between components.

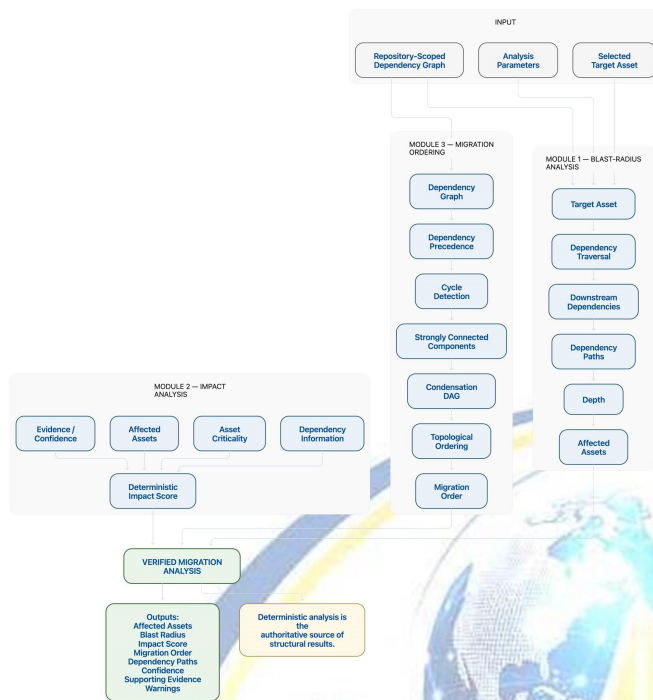


Fig. 6. Deterministic migration-impact analysis in CRYPTOMAP, comprising blast-radius analysis, impact analysis, and dependency-aware migration ordering.

E. Confidence and Provenance

Evidence is classified as Strong, Medium, or Weak according to implemented evidence rules. Direct API usage, explicit certificate/key configurations, and PEM blocks receive stronger classifications, while generic algorithm references receive weaker classifications. Asset confidence can be adjusted using bounded corroboration when evidence originates from multiple files or evidence types. Assets supported only by weak evidence are capped at a confidence of 0.69.

The resulting verified context contains the target asset, affected dependencies, graph relationships, migration sequence, impact metrics, and supporting provenance. This representation forms the deterministic output boundary for the planned local LLM.

VI. IMPLEMENTATION AND EXPERIMENTAL METHODOLOGY

A. Implementation

CRYPTOMAP was implemented as a modular backend pipeline using Python, FastAPI, PostgreSQL, Neo4j, and n8n. The repository scanner performs read-only traversal with directory, file-extension, and file-size filtering before passing relevant files to the cryptographic discovery engine.

The discovery engine applies deterministic pattern-based analysis to identify cryptographic algorithms, libraries, certificates, keys, TLS configurations, keystores, truststores, and HSM references. Detected evidence is represented with file

paths, line numbers, matched text, cryptographic concepts, evidence types, algorithms, confidence, and source information.

The extracted evidence is normalized into repository-scoped cryptographic assets and stored in Neo4j. The graph contains six node types—Certificate, Key, Application, Server, API, and HSM—and seven relationship types representing usage, dependency, invocation, hosting, signing, storage, and protection relationships. Repository identifiers are incorporated into graph identities and database constraints to maintain repository isolation.

The deterministic analysis layer provides blast-radius calculation, impact scoring, migration ordering, and cycle handling. PostgreSQL stores repository metadata, scan jobs, analysis results, progress information, and execution status. FastAPI exposes REST endpoints for repository management, asynchronous scan jobs, asset inspection, analysis operations, and verified-context generation. n8n provides external workflow orchestration through repository registration, job submission, status polling, and result retrieval.

The current implementation constructs a bounded verified context containing the analyzed target, affected assets, relationships, migration sequence, impact metrics, and provenance records. Local Ollama-based explanation and the grounding validator are planned components and are not included as implemented experimental results.

B. Experimental Setup

The implementation was evaluated in a local containerized environment using Python 3.12. PostgreSQL 16 and Neo4j 5.18 were deployed through Docker Compose. The evaluation focused on functional correctness of the implemented pipeline rather than complete detection coverage for arbitrary enterprise repositories.

The test suite contains 151 automated unit, integration, API, database, graph-analysis, reduction, security, and n8n contract tests. The complete suite was executed using:

```
./venv/bin/python -m pytest -q
```

The resulting baseline was 151 passed, 0 failed, and 0 skipped, with five non-blocking framework warnings and an execution time of approximately 9.15 seconds.

C. Evaluation Strategy

Cryptographic discovery and evidence extraction were tested using controlled repository examples containing supported source-code and configuration patterns. Evidence normalization and graph construction were evaluated through tests covering asset identity, deduplication, repository scoping, relationship construction, and graph constraints.

Deterministic migration analysis was tested through downstream dependency traversal, depth calculation, impact-score computation, migration precedence, and cyclic dependency handling through SCC condensation and DAG traversal. Confidence and provenance tests evaluated evidence classification, corroboration behavior, weak-evidence limits, and deterministic confidence calculation.

The reduction layer was evaluated for bounded-context construction and truncation behavior. API and n8n contract tests verified integration among job submission, asynchronous execution, status polling, and result retrieval. The evaluation therefore establishes implementation-level correctness of tested software contracts. It does not establish universal cryptographic detection accuracy, completeness across unsupported programming languages or native APIs, or real-world enterprise migration effectiveness.

VII. RESULTS, DISCUSSION AND LIMITATIONS

A. Results

The implemented **CRYPTOMAP** pipeline achieved 151/151 passing automated tests, with no failures or skipped cases and an execution time of approximately 9.15 seconds. These results verify the implemented software contracts across major system components.

The evaluation confirms functional support for progressive repository scanning, cryptographic evidence extraction, evidence normalization, repository-scoped graph construction, deterministic blast-radius analysis, impact scoring, migration ordering, SCC-based cycle handling, confidence and provenance evaluation, bounded verified-context construction, PostgreSQL job persistence, FastAPI services, and n8n orchestration.

The analysis layer produces reproducible outputs for the same graph and input parameters because traversal, scoring, confidence evaluation, and migration ordering use deterministic rules. The impact score incorporates criticality, confidence, asset type, and dependency depth, while migration ordering applies explicit precedence rules and deterministic tie-breaking.

B. Discussion

The implementation demonstrates an architecture in which authoritative migration analysis remains separate from generative reasoning. The current system can determine dependency scope, impact metrics, and migration order directly from graph data without requiring an LLM to infer these structural properties.

The progressive reduction mechanism limits information passed toward downstream reasoning by selecting relevant evidence and graph substructures rather than exposing an entire repository. The resulting verified context contains structural analysis and supporting provenance required for subsequent explanation.

The current results should be interpreted as implementation-validation results rather than empirical evidence of enterprise-wide effectiveness. Passing automated tests demonstrates that implemented behavior satisfies tested software contracts; it does not establish that every cryptographic dependency in an arbitrary repository will be detected or modeled correctly.

C. Limitations and Threats to Validity

The current discovery engine relies primarily on deterministic pattern-based analysis and therefore has incomplete

coverage for several cryptographic implementation mechanisms. Native C/C++ OpenSSL APIs, Java JCA/JCE calls, Go native cryptographic APIs, Rust cryptographic libraries, native post-quantum implementations, and compiled-binary reverse engineering are not currently supported.

The graph also depends on evidence available to the scanner. Dynamically generated configurations, undocumented dependencies, unavailable external services, and cryptographic behavior outside the analyzed repository may not be represented. Consequently, generated blast radius and migration order are bounded by the discovered graph.

The implemented impact score is a deterministic prioritization metric defined by selected weights and attenuation rules; it has not been calibrated against real enterprise migration outcomes. The planned Ollama-based explanation layer and grounding validator have not yet been experimentally evaluated.

The current evaluation uses automated tests and controlled repository scenarios rather than a large benchmark of real enterprise repositories. Broader evaluation should measure detection coverage, false positives, false negatives, graph completeness, scalability, and usefulness of migration recommendations across diverse repositories.

VIII. CONCLUSION AND FUTURE WORK

CRYPTOMAP presents an evidence-driven approach for analyzing post-quantum cryptography migration dependencies in software repositories. The implemented framework combines progressive repository scanning, deterministic cryptographic discovery, evidence normalization, dependency-graph construction, and rule-based migration analysis. Neo4j is used to represent relationships among migration-relevant assets, while deterministic algorithms calculate downstream blast radius, impact scores, and dependency-aware migration ordering. A central design principle of **CRYPTOMAP** is the separation of authoritative structural analysis from generative AI reasoning. The current implementation produces a verified and bounded analysis context containing affected assets, dependency relationships, migration order, impact information, confidence, and provenance. This allows future AI components to operate on a constrained representation instead of independently determining graph facts.

Automated testing provides implementation-level validation of the major implemented components and integration contracts. At the same time, the current discovery and evaluation limitations mean that these results should not be interpreted as evidence of complete cryptographic detection or enterprise-wide migration effectiveness.

Future work will extend cryptographic discovery to additional language-specific APIs and native implementations, evaluate **CRYPTOMAP** on larger and more diverse repositories, and measure detection coverage, false positives, false negatives, graph completeness, and scalability. The next stage will also integrate a local Ollama-based LLM for explanation generation and migration runbook creation, together with a

grounding validator that checks generated claims against the verified analysis context.

REFERENCES

- [1] K. F. Hasan, L. Simpson, M. A. R. Bae, C. Islam, Z. Rahman, W. Armstrong, P. Gauravaram, and M. McKague, "A Framework for Migrating to Post-Quantum Cryptography: Security Dependency Analysis and Case Studies," *IEEE Access*, vol. 12, pp. 23427–23450, 2024, doi: 10.1109/ACCESS.2024.3360412.
- [2] Y. Baseri, V. Chouhan, A. Ghorbani, and A. Chow, "Evaluation Framework for Quantum Security Risk Assessment: A Comprehensive Strategy for Quantum-Safe Transition," *Computers & Security*, vol. 150, Art. no. 104272, 2025.
- [3] E. Hirsch and K. Raab, "Architecture-Derived CBOMs for Cryptographic Migration: A Security-Aware Architecture Tradeoff Method," arXiv:2603.22442, 2026, doi: 10.48550/arXiv.2603.22442.
- [4] A. Adebisin, "Quantum-aware Software Engineering: An AI-driven Framework for Measuring Cryptographic Agility in Enterprise Systems," *GSC Advanced Research and Reviews*, vol. 25, no. 3, pp. 374–384, 2025, doi: 10.30574/gscarr.2025.25.3.0379.
- [5] S. Kru"ger, S. Nadi, M. Reif, K. Ali, M. Mezini, E. Bodden, F. Go"pfert, F. Gu"nther, C. Weinert, D. Demmler, and R. Kamath, "CogniCrypt: Supporting Developers in Using Cryptography," in *Proc. 32nd IEEE/ACM Int. Conf. Automated Software Engineering (ASE)*, pp. 931–936, 2017, doi: 10.1109/ASE.2017.8115707.
- [6] S. Rahaman, Y. Xiao, S. Afrose, F. Shaon, K. Tian, M. Frantz, D. Yao, and M. Kantarcioglu, "CryptoGuard: High Precision Detection of Cryptographic Vulnerabilities in Massive-sized Java Projects," arXiv:1806.06881, 2018.
- [7] M. Schlichtig, A.-K. Wickert, S. Kru"ger, E. Bodden, and M. Mezini, "CamBench – Cryptographic API Misuse Detection Tool Benchmark Suite," arXiv:2204.06447, 2022.
- [8] E. Hirsch, K. Raab, T. J. Bauer, and D. Loebenberg, "Detecting Cryptographically Relevant Software Packages with Collaborative LLMs," in *Proc. 12th Int. Conf. Information Systems Security and Privacy (ICISSP)*, vol. 2, pp. 354–365, 2026, doi: 10.5220/0014441400004061.
- [9] C. Fritz, C.-P. Georg, A. Mele, and M. Schweinberger, "Vulnerability Webs: Systemic Risk in Software Networks," arXiv:2402.13375, 2024.
- [10] L. Zhang, C. Liu, F. Zhang, Y. Zhang, Y. Zhou, and Y. Liu, "Implicit, Yet Impactful: Understanding Hidden Dependencies in Java Projects," arXiv:2608.16262, 2026.
- [11] T. Sharma, P. Singh, and D. Spinellis, "An Empirical Investigation on the Relationship Between Design and Architecture Smells," *Empirical Software Engineering*, vol. 25, no. 5, pp. 4020–4068, 2020, doi: 10.1007/s10664-020-09847-2.
- [12] I. Pashchenko, H. Plate, S. E. Ponta, A. Sabetta, and F. Massacci, "Vulnerable Open Source Dependencies: Counting Those That Matter," in *Proc. Int. Symp. Empirical Software Engineering and Measurement (ESEM)*, 2018, doi: 10.1145/3239235.3268920.
- [13] C. Fang et al., "Large Language Models for Code Analysis: Do LLMs Really Do Their Job?," in *USENIX Security Symposium*, 2024.
- [14] K. Wang et al., "SGCR: A Specification-Grounded Framework for Trustworthy LLM Code Review," arXiv:2512.17540, 2025, doi: 10.48550/arXiv.2512.17540.
- [15] S. M. Abtahi and A. Azim, "Augmenting Large Language Models with Static Code Analysis for Automated Code Quality Improvements," arXiv:2506.10330, 2025.
- [16] V.-V. Nguyen and T. V. Nguyen, "Large Language Models in Software Engineering: A Systematic Review and Vision," *Journal of Education for Sustainable Innovation*, vol. 2, no. 2, pp. 146–156, 2024, doi: 10.56916/jesi.v2i2.968.
- [17] W. Newhouse et al., "Migration to Post-Quantum Cryptography: Preparation for Considering the Implementation and Adoption of Quantum Safe Cryptography," NIST SP 1800-38, 2023.
- [18] CISA, NSA, and NIST, "Quantum Readiness: Migration to Post-Quantum Cryptography," Joint Cybersecurity Information / Fact Sheet, 2023.
- [19] Cloud Security Alliance, "Post-Quantum Cryptography Key Management," 2026.
- [20] W. Newhouse et al., "Migration to Post-Quantum Cryptography: Quantum Readiness: Cryptographic Discovery – Volume B: Approach, Architecture, and Security Characteristics of Public Key Application Discovery Tools," NIST SP 1800-38B, 2023.