BF-CBOM: Uncovering Cryptographic Assets Through Comparative CBOM Analysis at Scale

Roman Bögli
University of Bern
Bern, Switzerland
roman.boegli@unibe.ch

Jonas Spieler
University of Bern
Bern, Switzerland
jonas.spieler@unibe.ch

Timo Kehrer
University of Bern
Bern, Switzerland
timo.kehrer@unibe.ch

Abstract

The advancing threat of quantum-capable adversaries accelerates the need to locate and replace vulnerable cryptographic assets in software systems. To support this transition, Cryptography Bills of Materials (CBOMs) are becoming essential for inventorying the cryptographic footprint of software systems as increasingly demanded by regulators in critical domains. While first CBOM generation tools have emerged, they still lack reliable means to comprehend and analyze the cryptographic landscape of codebases.

We present BF-CBOM, a first-of-its-kind framework for orchestrating various CBOM generators and analyzing their outputs, enabling holistic comprehension of the cryptographic posture of software projects. BF-CBOM offers a containerized environment designed to accommodate the heterogeneous toolchains of such generators, executes them on GitHub code repositories, and aggregates their outputs for comparative investigation in a unified analysis layer. Our preliminary study reveals striking discrepancies between generated CBOMs, underscoring the need for systematic evaluation. BF-CBOM supports researchers with cryptographic reports, practitioners through CI/CD integration, and tool developers by providing performance feedback relative to other CBOM generators. A demonstration video is available at youtu.be/-YdBPHsyymU.

Keywords

CBOM, software analysis, cryptography, post-quantum, security

ACM Reference Format:
Roman Bögli, Jonas Spieler, and Timo Kehrer. 2026. BF-CBOM: Uncovering Cryptographic Assets Through Comparative CBOM Analysis at Scale. In *34th IEEE/ACM International Conference on Program Comprehension (ICPC '26), April 12–13, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3794763.3794831

1 Introduction

The security of modern software systems critically depends on a thorough understanding of their components, including their origins, configurations, and interdependencies. Without such transparency, organizations become vulnerable to supply chain attacks that exploit overlooked or poorly documented components [17, 48]. Software Bills of Materials (SBOMs) [33, 49] have therefore become a widely established means to increase supply-chain security, providing inventories of software dependencies. With the rise of quantum computing, however, these mitigation measures demand even greater attention. When sufficiently powerful quantum computers become available, they could rapidly render current, classical cryptographic algorithms insecure [5, 14] using long-known quantum algorithms [40, 41]. In response to this disruption, the NIST Post-Quantum Cryptography (PQC) standardization project has defined new algorithmic standards as of August 2024 [22].

However, new quantum-secure standards are not enough. The focus now shifts to organizations, which must identify and assess their cryptographic posture across their software landscape to plan appropriate action [18, 19, 21]. Systems must also remain adaptable to evolving cryptographic needs, a property known as *cryptographic agility* [25, 27, 28]. To achieve crypto-agility at scale, organizations require an interoperable, machine-readable inventory of cryptographically relevant information that can be automatically generated and audited [36]. At the heart of this effort are *Cryptography Bills of Materials* (CBOMs) [8, 26] – structured object models that describe cryptographic components and their dependencies within software systems. CBOMs extend SBOMs to the cryptographic domain, providing foundational inventory capabilities necessary for PQC migration, i.e., the assessment and systematic replacement or augmentation of classical cryptographic primitives in software systems with quantum-resistant alternatives.

Yet producing high-quality CBOMs is substantially more difficult than generating SBOMs. Cryptography-related information is scattered across code and third-party libraries, often hidden in configuration files or runtime artifacts such as certificates, keys, and protocol settings — all frequently undocumented or context-dependent. Although first CBOM generators [7, 30, 35] and standardization efforts such as CycloneDX [26] have emerged, their reliability and coverage remain entirely unexplored. No systematic approach exists to assess how comprehensively these generators detect and report cryptographic assets. Moreover, given the immaturity of individual tools, any single generator's output provides only a fragmented view of a codebase's cryptographic posture. This leaves both researchers and practitioners without a sound basis for trust or adoption. To close this gap, we present BF-CBOM, a first-of-its-kind framework for uncovering cryptographic assets at scale through orchestrated CBOM generation and comparative output analysis. BF-CBOM provides a controlled and reproducible environment for integrating heterogeneous CBOM generators to inspect a given codebase and collect their outputs for uniform analysis and exploration. Additionally, the framework supports batch inspections across sampled GitHub repositories, laying the foundation for systematic benchmarking of CBOM generator capabilities. The framework is modular and extensible: CBOM generators with

ICPC '26, Rio de Janeiro, Brazil

ACM ISBN 979-8-4007-2482-4/2026/04
https://doi.org/10.1145/3794763.3794831

diverse technology stacks can be containerized, deployed, and evaluated side by side. This enables researchers to assess the strengths and weaknesses of existing approaches and helps practitioners gain holistic visibility into their codebases' cryptographic postures. By empowering users to combine multiple generators in a low-effort, programmable, and repeatable ensemble, BF-CBOM is not designed to resolve discrepancies but to actively surface them – helping comprehend tool adoption trade-offs and increasing transparency in a space where overstated capabilities can have serious consequences. In summary, this paper makes the following contributions:

- We introduce BF-CBOM, a first-of-its-kind framework for inspecting software's cryptographic posture at scale by orchestrating multiple CBOM generators and comparatively analyzing their outputs.
- We integrate three real-world CBOM generators and an additional LLM-based approach into BF-CBOM, delivering a functional, ready-to-use prototype.
- We present first insights gained from initial experiments using BF-CBOM, demonstrating the discrepancies between generated CBOMs and the need for systematic evaluation.
- We discuss the envisioned application areas of BF-CBOM for researchers and practitioners, outlining its potential to augment the field of cryptographic supply chain security.

As accompanying artifacts to this paper, we provide a replication package consisting of BF-CBOM [4] and a demonstration video [3].

2 Background & Related Work

BOM Standards. The concept of a Bill of Materials (BOM) originates from manufacturing, where it denotes a structured inventory of raw materials and components required to produce a product, and has been recently adopted in software supply chain management [43]. Standardization efforts are predominantly driven by two initiatives, the Linux Foundation's Software Package Data Exchange (SPDX) [46] and the OWASP Foundation's CycloneDX BOM Specification (ECMA-424) [8]. While the more established concept of a Software Bill of Materials (SBOM) captures software components and their dependencies on a coarse-grained level, the more recently introduced concept of a Cryptography Bill of Materials (CBOM) aims to inventory the usage of *cryptographic assets* within a software system. In April 2024, CycloneDX v1.6 [26] extended previous versions of the standard by first-class objects for cryptographic algorithms, keys, certificates, and protocol settings. These fine-grained extensions shall support reasoning about correctness, compliance, and post-quantum risk beyond SBOMs.

CBOM Generation Tools. To the best of our knowledge, only three tools currently support automated CBOM generation, all of which report CBOMs according to the CycloneDX standard and output them in JSON format: (i) `CBOMKit` [30] is a dedicated CBOM generator for Python and Java projects, originally developed by IBM Research in Sept. 2024 and now maintained by the Post-Quantum Cryptography Alliance [32]. It leverages the commercial code analysis tool SonarQube [42] to identify cryptographic assets, and generates CBOMs that can be enriched through configurable compliance policy checks. Beyond its core generation engine (backend) which provides an API for programmatic access, `CBOMKit` ships with a database for persisting CBOMs and a web-based GUI for interactive exploration of the results; (ii) `cdxgen` [7] is the official generation tool by CycloneDX. Though primarily focused on generating SBOMs, since April 2024 (v11.7.0), the generated SBOMs can optionally be extended with CBOM information. As in `CBOMKit`, however, language support for this extension is limited to Python and Java projects; (iii) `CryptobomForge` [35] is a CLI tool developed and maintained by Santander Security Research and, like `CBOMKit`, focuses exclusively on CBOM generation and rule-based cryptographic compliance analysis. Its first version was released in Dec. 2023, making it the oldest of the three CBOM generators. `CryptobomForge` does not analyze source code directly but processes CodeQL [15] analysis outputs (SARIF files [24]) as input.

Academic Studies. The increasing amount and severity of software supply chain attacks have motivated several studies on SBOM maturity [49]. For example, Sehgal et al. [39] survey SBOM generation approaches, highlighting tooling fragmentation and the need for systematic evaluation. Comparison studies reveal that current SBOM generation tools provide only partial insights [45], and differential analyses uncover widespread inconsistencies [50]. Ecosystem-focused studies expose issues such as incorrect versions, remote dependencies, and missing metadata [6]. Security analyses reveal the limitations of current SBOM-based vulnerability detection and propose more accurate generators [1]. Adoption studies show growing but incomplete uptake in open-source projects, with many SBOMs lacking required fields or version control [23]. Broader reviews of supply chain threats further stress the need for robust tooling [48]. In sum, these studies highlight progress and persistent challenges in the SBOM tooling landscape, and emphasize the need for comparative evaluation and benchmarking to drive maturity.

As opposed to SBOMs, the younger concept of CBOMs has received considerably less interest so far [10, 16], with no academic studies to date. Building on lessons from the SBOM field, however, we argue that comparative studies of CBOM generation are essential, as cryptographic assets play a pivotal role in the upcoming PQC migration. Our CBOM-first perspective addresses this gap as a prerequisite for secure and reliable adoption.

3 Inspection Workflow

We now introduce our inspection workflow with BF-CBOM, designed to systematically scan code repositories using the different CBOM generation tools available to date and compare their output.

Setup. To begin, one configures a new inspection by defining the GitHub repositories to be analyzed and selecting the CBOM generation approaches to be used. BF-CBOM offers inspecting either (i) a single repository or (ii) batch-processing a set of repositories, sampled randomly or from a user-defined list of URLs. While the former addresses repo-specific program comprehension, the latter lays the foundation for large-scale, systematic benchmarking.

Execution. Launching an inspection triggers all CBOM generation jobs in the background, dispatched to the respective integrated tools (detailed in Sec. 4.1). The workflow allows monitoring these jobs through an interactive overview that visualizes individual progress. Worker feedback on job states, such as *pending*, *completed successfully*, *failed*, or *timed out*, continuously informs the inspection process. For each completed job, we record both the generated CBOM and the total runtime required to generate it.

Analysis. The workflow concludes with the analysis layer, offering a multi-perspective view on the consolidated inspection results. Beyond descriptive job outcome statistics, it correlates runtimes with repository sizes (KB) and the number of reported components. The latter correlation is particularly valuable, as it reveals potential over- or under-reporting patterns across different CBOM generation approaches. Given the significance of component-level analysis, BF-CBOM offers drill-down capabilities for direct examination of individual CBOM fragments. Crucially, its component matching feature (detailed in Sec. 4.2) identifies overlapping discoveries, revealing which components were consistently detected across generation approaches for a given repository. By aggregating findings from multiple generators, this collective view deepens comprehension of cryptographic asset usage and provides interactive access to the underlying raw CBOM excerpts.

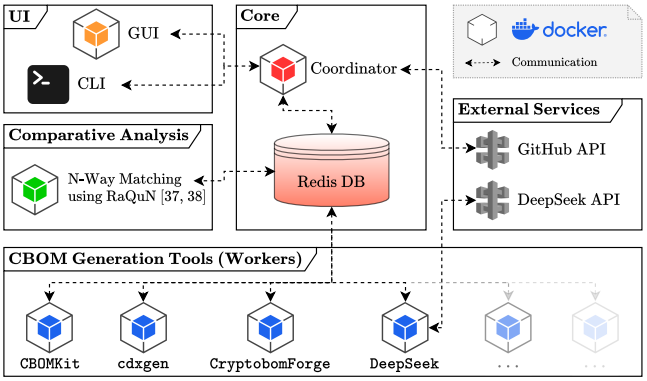

Figure 1: BF-CBOM's System Architecture.

4 Architecture & Features

The architecture of BF-CBOM is deliberately modular and relies on Docker Compose [13] to orchestrate its components. It combines a central coordination engine, an integration interface for the CBOM generation strategies under evaluation, and a module for subsequent comparative analysis. Figure 1 illustrates this design by providing a conceptual overview of the system's modular architecture, grouping logically related components into named boxes.

Core. The core component of BF-CBOM is the Coordinator, orchestrating communication between all other components. We employ a message-driven architecture for this coordination, leveraging Redis [31] both as a database for persisting intermediate results and as a streaming platform supporting the publish-subscribe pattern. This design enables asynchronous, scalable, and decoupled component interactions, making the system flexible for extension.

UI. The user interface is supported through two complementary entry points: a web-based Streamlit [44] GUI for interactive inspection management and visualization, and a CLI that provides programmatic access for automation scenarios or CI/CD integration. Both interfaces forward requests to the Coordinator, which then dispatches tasks to the appropriate job queue.

CBOM Generation Tool (Worker). Each CBOM generation tool runs in its own containerized environment, ensuring reproducibility and portability. This allows BF-CBOM to accommodate not only standalone user-facing tools but also alternative CBOM generation approaches involving scripts, libraries, or LLM-based methods. We refer to all such approaches as *workers*, each operating independently with its dependencies. Communication is mediated by a narrow-waist messaging interface that translates between the Coordinator's job dispatching and a worker's execution. This design highlights BF-CBOM's extensibility, as new workers can be uniformly integrated through this interface regardless of their specialized toolchain, underlying architecture, or internal complexity. Details on the integration of the four workers follow in Sec. 4.1.

Comparative Analysis. The comparative analysis component provides functionality for in-depth inspection of harvested CBOMs once an inspection has completed, going beyond basic statistics and visualizations. At present, the tool identifies commonalities and differences across reported cryptographic assets recorded by different workers, a non-trivial task addressed in detail in Sec. 4.2.

External Services. Some BF-CBOM functionality requires interfacing with external services. The GitHub API, for example, is used by the Coordinator to sample repositories and enrich them with metadata. Alternatively, repository URLs can be supplied directly, allowing integration with external sampling frameworks [9]. Likewise, some CBOM generators (workers) may themselves require outbound communication.

4.1 Integrated CBOM Generation Tools

In its current version, our BF-CBOM prototype integrates four distinct CBOM generation tools, each embedded in its own worker.

The **`CBOMKit`** worker runs `CBOMKit` v2.1.10 [30] in headless mode and communicates with its backend API via WebSocket [20]. Repository cloning is handled by `CBOMKit` itself prior to CBOM generation. The worker retrieves the generated CBOM upon completion through the same communication channel. The **`cdxgen`** worker accommodates `cdxgen` 1.7.0 [7], using its native CLI tool directly on the previously cloned repositories and reading the generated BOM files from disk. The **`CryptobomForge`** worker operates `CryptobomForge` v1.1.0 [35] and first runs CodeQL analyses using the CodeQL CLI on the previously cloned repository. The resulting SARIF file [24] is then processed by `CryptobomForge` to generate the CBOM, which the worker reads from disk for return. Among all workers, `CryptobomForge` has the highest setup overhead owing to the complexity of the heavy CodeQL toolchain [15].

With the fourth and last worker **`DeepSeek`**, we deliberately pursued an alternative CBOM generation approach using the general-purpose Large Language Model (LLM) DeepSeek [11] through its official API [12]. The rationale behind this is twofold. First and foremost, to showcase BF-CBOM's extensibility, as any CBOM generation approach can be integrated through its Docker-based environment and the narrow-waist messaging interface that connects it to the coordination engine. Second, to explore the potential of off-the-shelf, zero-shot LLM methods for detecting cryptographic usage via prompt-based code analysis. Specifically, DeepSeek receives a repository URL and is instructed to generate a standards-compliant CBOM. We thereby lay the groundwork for future systematic evaluations, as we view the repurposing of LLM-based code analysis to extract cryptographic assets and report them in a CBOM-compliant manner as a promising emerging direction in this space.

4.2 Comparative Analysis of Crypto-Assets

Central to comprehending a codebase's cryptographic posture is BF-CBOM's comparative analysis, which identifies commonalities and differences among the CBOMs produced by different workers. A pilot study revealed that our supported CBOM generation approaches create JSON documents whose structure and level of detail differ substantially (detailed in Sec. 5). Consequently, simple text-based comparison tools produced mostly low-level noise rather than meaningful insights. To address this, we lifted the analysis to a conceptual level, focusing on identifying the common and unique cryptographic assets reported in generated CBOMs.

Technically, each cryptographic asset is represented by a JSON component (a subtree in the document), and corresponding components must be identified across the different CBOMs. Conceptually, this task corresponds to an n-way matching problem [34], where n is the number of input JSON documents. Since n-way matching is NP-hard in general, we adopted a heuristic approach inspired by prior work on variability mining for software product line extraction [2, 29]. To this end, BF-CBOM currently employs RaQuN [37, 38], a configurable n-way matcher that identifies matching candidates through clustering. In RaQuN, conceptual elements are mapped into a vector space that supports efficient range queries in a high-dimensional search tree. To adapt RaQuN to our context, extracted CBOM components are vectorized by selecting the most relevant fields from each JSON component and embedding them using the language model MiniLM-v2 [47], which offers an adequate trade-off between inference speed and output quality.

5 Insights from First Experiments

As this work instroduces BF-CBOM as a novel framework, we reserve systematic validation and a broader usability study for a subsequent full research paper. Nevertheless, conversations with two domain experts underscored the practical relevance of comparatively analyzing CBOMs produced by different generation approaches to obtain a more comprehensive view of a software system's cryptographic footprint. We therefore deem it valuable to present first, explorative findings from initial experiments using our prototype implementation [4].

As a single entry point, BF-CBOM abstracts away the heterogeneous toolchains of CBOM generators, simplifying cryptographic asset discovery by aggregating insights from multiple tools into a unified view. First batch inspections revealed considerable variation in tool behavior. Runtimes ranged from seconds to several minutes depending on repository size, raising concerns for CI/CD integration where predictable performance is critical. Stability issues were also common, with some tools failing on specific repositories, revealing fragile toolchains and heavy dependency footprints. These observations show that batch inspections must assess not only correctness but also robustness and operational feasibility.

Reported cryptographic assets were strikingly inconsistent. Tools disagreed on which cryptographic components to include and on how much metadata to provide. Overlaps were rare, with almost no lexically exact matches. While our n-way matching strategy still uncovered meaningful correspondences, such discrepancies illustrate misalignment in CBOM practices and the need for systematic comparison to build confidence in generated inventories.

In sum, our explorative findings confirm the immaturity of CBOM tooling – reinforcing the value of investigating code with multiple generators in ensemble to compensate for individual blind spots. We see BF-CBOM as an early yet pivotal instrument for surfacing concrete evidence of that immaturity. Yet, even at prototype stage, it enables findings that would be tedious to obtain manually.

6 Conclusion and Future Work

With this paper, we introduced BF-CBOM [3, 4], a first-of-its-kind framework for inspecting code repositories' cryptographic posture through multiple CBOM generator tools. In this prototype, we integrated three productive CBOM generators plus one LLM-based approach, demonstrating that BF-CBOM's containerized architecture can host heterogeneous tools behind a uniform, message-driven interface. In a three-step workflow, users configure individual or batch inspections, execute the integrated CBOM generators, and compare the resulting CBOMs within a unified analysis layer.

Envisioned Application Areas. We see valuable utility of BF-CBOM in the following areas: *(i) Practitioners* gain direct support in daily activities. Software developers can integrate the framework into CI/CD pipelines and (batch) inspect CBOMs from multiple generators at once, supporting the comprehension of their projects' underlying cryptographic assets. CBOM vendors can benchmark and refine their own CBOM generators by comparing them against competing tools. Standards designers can use it to validate and improve emerging CBOM specifications, revealing ambiguities and misunderstandings. Compliance officers can assess organizational cryptographic posture across system landscapes and streamline audits and reporting, particularly during PQC migration. *(ii) Researchers* gain an instrument for advancing the field. Primarily, it enables evaluating the effectiveness of emerging CBOM generators and comparing them to each other, allowing thorough studies that expand on our initial insights. We deem this particularly valuable for large-scale empirical studies as BF-CBOM's architecture is easy to parallelize. Security researchers can use it to more holistically analyze cryptographic usage and post-quantum readiness across software ecosystems and libraries in the wild.

Future Work. We plan to integrate further CBOM generation approaches as they evolve and extend support for alternative reporting formats (e.g., XML, YAML) with enhanced schema validation for rigorous standards compliance. To extend benchmarking beyond relative comparisons, we are curating ground-truth datasets with manually verified cryptographic inventories. We further envision adapting BF-CBOM's concept to other BOM domains where exposing cross-generator overlaps and discrepancies can further strengthen confidence in reported components.

With BF-CBOM as a foundational instrument for navigating the emerging CBOM landscape, we believe it will accelerate the development of CBOM practices, promote their broader adoption, and help organizations comprehend their software's cryptographic footprint to streamline the post-quantum transition.

Acknowledgments

We thank Christian Cachin from the University of Bern (*Cryptology & Data Security Group*) and Marc Stöcklin from IBM Research Zurich (*Security Department*) for their valuable feedback on this work.

References

[1] Giacomo Benedetti, Serena Cofano, Alessandro Brighente, and Mauro Conti. 2025. The Impact of SBOM Generators on Vulnerability Assessment in Python: A Comparison and a Novel Approach. In *Proc. Int'l Conf. on Applied Cryptography and Network Security (ACNS)*. Vol. 15826. Springer, 487–509. doi:10.1007/978-3-031-95764-2_19.

[2] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wąsowski. 2013. A survey of variability modeling in industrial practice. In *Proc. Int'l Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*. ACM, 7:1–7:8. doi:10.1145/2430502.2430513.

[3] Roman Bögli. BF-CBOM: Your Best Friend for Cryptography Bill of Materials (DEMO). YouTube, (Dec. 7, 2025). https://youtu.be/-YdBPHsyymU.

[4] [SW] Roman Bögli, Jonas Spieler, and Timo Kehrer, BF-CBOM: Your Best Friend for Generating, Understanding, and Comparing Cryptography Bill of Material (CBOMs) version v1.0.3, Dec. 2025. doi:10.5281/zenodo.17853399, URL: https://github.com/SEG-UNIBE/BF-CBOM.

[5] Gilles Brassard. 1994. Quantum computing: the end of classical cryptography? *SIGACT News*, 25, 4, 15–21. doi:10.1145/190616.190617.

[6] Serena Cofano, Giacomo Benedetti, and Matteo Dell'Amico. 2024. SBOM Generation Tools in the Python Ecosystem: an In-Detail Analysis. In *Proc. Int'l Conf. on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, 427–434. doi:10.1109/TrustCom63139.2024.00077.

[7] [SW] CycloneDX, Cdxgen version 11.7.0, Sept. 2, 2025. URL: https://github.com/CycloneDX/cdxgen.

[8] CycloneDX. 2024. CycloneDX Bill of Materials Specification (ECMA-424). Version v1.6. (Apr. 9, 2024). https://github.com/CycloneDX/specification.

[9] Ozren Dabic, Rosalia Tufano, and Gabriele Bavota. 2024. SEART Data Hub: Streamlining Large-Scale Source Code Mining and Pre-Processing. In *Proc. Int'l Conf. on Software Maintenance and Evolution (ICSME)*. IEEE, 888–892. doi:10.1109/ICSME58944.2024.00097.

[10] dblp.org. 2025. SPARQL Query (CBOM- vs. SBOM-related publications). Retrieved Dec. 7, 2025 from https://web.archive.org/web/20251207133728/https://sparql.dblp.org/lyS1l5.

[11] DeepSeek. 2025. Retrieved Sept. 23, 2025 from https://www.deepseek.com/.

[12] DeepSeek API Platform. 2025. Retrieved Sept. 23, 2025 from https://platform.deepseek.com/.

[13] [SW], Docker/Compose version v2.39.2, Aug. 8, 2025. URL: https://github.com/docker/compose.

[14] Pete Ford. 2023. The Quantum Cybersecurity Threat May Arrive Sooner Than You Think. *Computer*, 56, 2, 134–136. doi:10.1109/MC.2022.3227657.

[15] GitHub. 2025. CodeQL. Retrieved Sept. 24, 2025 from https://codeql.github.com.

[16] Google Trends. 2025. SBOM vs. CBOM vs. Post-Quantum Cryptography (Jan'21-Nov'25). Retrieved Dec. 6, 2025 from https://trends.google.com/trends/explore?date=2021-01-01%202025-12-01&q=SBOM,CBOM,%2Fm%2F0bhc70l.

[17] Sivana Hamer, Jacob Bowen, Md. Nazmul Haque, Robert Hines, Chris Madden, and Laurie A. Williams. 2025. Closing the Chain: How to reduce your risk of being SolarWinds, Log4j, or XZ Utils. *Computing Research Repository (CoRR)*. doi:10.48550/ARXIV.2503.12192.

[18] David Joseph et al. 2022. Transitioning organizations to post-quantum cryptography. *Nature*, 605, 7909, 237–243. doi:10.1038/S41586-022-04623-2.

[19] Nils Lohmiller, Sabrina Kaniewski, Michael Menth, and Tobias Heer. 2025. A Survey of Post-Quantum Cryptography Migration in Vehicles. *IEEE Access*, 13, 10160–10176. doi:10.1109/ACCESS.2025.3528562.

[20] Alexey Melnikov and Ian Fette. 2011. The WebSocket Protocol. RFC 6455. (Dec. 2011). doi:10.17487/RFC6455.

[21] Christian Näther, Daniel Herzinger, Stefan-Lukas Gazdag, Jan-Philipp Steghöfer, Simon Daum, and Daniel Loebenberger. 2024. Migrating Software Systems Toward Post-Quantum Cryptography-A Systematic Literature Review. *IEEE Access*, 12, 132107–132126. doi:10.1109/ACCESS.2024.3450306.

[22] National Institute of Standards and Technology (NIST). 2024. NIST Releases First 3 Finalized Post-Quantum Encryption Standards. *Press Release*, (Aug. 13, 2024). Retrieved May 7, 2025 from https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards.

[23] Sabato Nocera, Simone Romano, Massimiliano Di Penta, Rita Francese, and Giuseppe Scanniello. 2025. On the adoption of software bill of materials in open-source software projects. *J. Systems and Software (JSS)*, 230, 112540. doi:10.1016/J.JSS.2025.112540.

[24] OASIS. 2023. Static Analysis Results Interchange Format (SARIF). Version v2.1.0. (Aug. 28, 2023). https://docs.oasis-open.org/sarif/sarif/v2.1.0/sarif-v2.1.0.html.

[25] David Ott, Kenny Paterson, and Dennis Moreau. 2023. Where Is the Research on Cryptographic Transition and Agility? *Communications of the ACM*, 66, 4, 29–32. doi:10.1145/3567825.

[26] OWASP Foundation. 2024. Authoritative Guide to CBOM: Implement Cryptography Bill of Materials for Post-Quantum Systems and Applications. First Edition. CycloneDX Feature Working Group on Cryptography, (Apr. 9, 2024). Retrieved May 7, 2025 from https://cyclonedx.org/guides/OWASP_CycloneDX-Authoritative-Guide-to-CBOM-en.pdf.

[27] Sebastian Paul and Melanie Niethammer. 2019. On the importance of cryptographic agility for industrial automation. *at - Automatisierungstechnik*, 67, 5, 402–416. doi:10.1515/AUTO-2019-0019.

[28] Kyrylo Petrenko, Atefeh Mashatan, and Farid Shirazi. 2019. Assessing the quantum-resistant cryptographic agility of routing and switching IT network infrastructure in a large-size financial organization. *Journal of Information Security and Applications (JISA)*, 46, 151–163. doi:10.1016/J.JISA.2019.03.007.

[29] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. 2005. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer. doi:10.1007/3-540-28901-1.

[30] [SW] Post-Quantum Cryptography Alliance (PQCA), CBOMkit version 2.1.10, Aug. 7, 2025. URL: https://github.com/PQCA/cbomkit.

[31] [SW], Redis version 8.2.1, Aug. 2, 2025. URL: https://github.com/redis/redis.

[32] IBM Research. 2025. IBM is donating its CBOM toolset to the Linux Foundation. (June 23, 2025). Retrieved Sept. 24, 2025 from https://research.ibm.com/blog/cryptographic-cbom-linux-foundation.

[33] Dirk Riehle. 2025. The Software Bill of Materials. *Computer*, 58, 4, 115–120. doi:10.1109/MC.2025.3530276.

[34] Julia Rubin and Marsha Chechik. 2013. N-way model merging. In *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE)*. ACM, 301–311. doi:10.1145/2491411.2491446.

[35] [SW] Santander Security Research, Cryptobom-Forge version 1.1.0, Jan. 25, 2024. URL: https://github.com/Santandersecurityresearch/cryptobom-forge.

[36] Nicolai Schmitt, Johanna Henrich, Dominik Heinz, Nouri Alnahawi, and Alexander Wiesmaier. 2024. On Criteria and Tooling for Cryptographic Inventories. In *Sicherheit*. Vol. P345. Gesellschaft für Informatik e.V., 49–63. doi:10.18420/SICHERHEIT2024_003.

[37] Alexander Schultheiß, Paul Maximilian Bittner, Alexander Boll, Lars Grunske, Thomas Thüm, and Timo Kehrer. 2023. RaQuN: a generic and scalable n-way model matching algorithm. *Software and System Modeling (SoSyM)*, 22, 5, 1495–1517. doi:10.1007/s10270-022-01062-5.

[38] Alexander Schultheiß, Paul Maximilian Bittner, Lars Grunske, Thomas Thüm, and Timo Kehrer. 2021. Scalable N-Way Model Matching Using Multi-Dimensional Search Trees. In *Proc. Int'l Conf. on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 1–12. doi:10.1109/MODELS50736.2021.00010.

[39] Vandana Verma Sehgal and P. S. Ambili. 2024. A Taxonomy and Survey of Software Bill of Materials (SBOM) Generation Approaches. In *Analytics Global Conference (ACG) 2023*. Springer, 40–51. doi:10.1007/978-3-031-50815-8_3.

[40] P.W. Shor. 1994. Algorithms for quantum computation: discrete logarithms and factoring. In *Proc. Annual IEEE Symposium on Foundations of Computer Science*, 124–134. doi:10.1109/SFCS.1994.365700.

[41] Harbaksh Singh. 2024. Managing the Quantum Cybersecurity Threat: Harvest Now, Decrypt Later. In *Quantum Computing: A Journey into the Next Frontier of Information and Communication Security*. (1st ed.). CRC Press. Chap. 9, 142–158. ISBN: 978-1-003-47528-6. doi:10.1201/9781003475286.

[42] SonarSource. 2025. SonarQube. Retrieved Sept. 24, 2025 from https://www.sonarsource.com/.

[43] Trevor Stalnaker, Nathan Wintersgill, Oscar Chaparro, Massimiliano Di Penta, Daniel M. Germán, and Denys Poshyvanyk. 2024. BOMs Away! Inside the Minds of Stakeholders: A Comprehensive Study of Bills of Materials for Software Systems. In *Proc. Int'l Conf. on Software Engineering (ICSE)*. ACM, 44:1–44:13. doi:10.1145/3597503.3623347.

[44] [SW], Streamlit version 1.50.0, Sept. 23, 2025. URL: https://github.com/streamlit/streamlit.

[45] Lucas Tate, Rebecca Jones, Doug Dennis, Tatyana Benko, and Jody Askren. 2024. Comparing Bills of Materials. *Computing Research Repository (CoRR)*. doi:10.48550/ARXIV.2411.10384.

[46] The Linux Foundation. 2025. SPDX Specification v3.0.1. Retrieved Sept. 27, 2025 from https://spdx.github.io/spdx-spec/v3.0.1/.

[47] Wenhui Wang, Hangbo Bao, Shaohan Huang, Li Dong, and Furu Wei. 2021. MiniLMv2: Multi-Head Self-Attention Relation Distillation for Compressing Pretrained Transformers. In *Findings of the Association for Computational Linguistics (ACL/IJCNLP)*, 2140–2151. doi:10.18653/V1/2021.FINDINGS-ACL.188.

[48] Laurie Williams et al. 2025. Research Directions in Software Supply Chain Security. *Trans. on Software Engineering and Methodology (TOSEM)*, 34, 5, 1–38. doi:10.1145/3714464.

[49] Boming Xia, Tingting Bi, Zhenchang Xing, Qinghua Lu, and Liming Zhu. 2023. An Empirical Study on Software Bill of Materials: Where We Stand and the Road Ahead. In *Proc. Int'l Conf. on Software Engineering (ICSE)*, 2630–2642. doi:10.1109/ICSE48619.2023.00219.

[50] Sheng Yu, Wei Song, Xunchao Hu, and Heng Yin. 2024. On the Correctness of Metadata-Based SBOM Generation: A Differential Analysis Approach. In *Int'l Conf. on Dependable Systems and Networks (DSN)*. IEEE/IFIP, 29–36. doi:10.1109/DSN58291.2024.00018.