QED-Lite: Lightweight Detection of Quantum-Vulnerable ELF Binaries via Cryptographic Library Version Fingerprinting

Ha-Gyeong Kim[1 [0009−0008−5840−5517]], Seung-Won Lee[1 [0009−0009−9583−1387]], Ji-Won Bang[1 [0009−0006−5453−1722]], Ui-Jae Kim[1 [0009−0004−2617−9659]], Hui-Ju Kang[1 [0009−0008−3198−4598]], Min-Seo Kim[1 [0009−0008−1979−5778]], and Hwa-Jeong Seo[1 [0000−0003−0069−9061]]

Hansung University, Seoul, Korea
kimhaha4420@gmail.com, dkajdfhd1@gmail.com, bgjiiw0n754@gmail.com, neome0919@gmail.com, chrisvt424@gmail.com, msqoo0212@gmail.com, hwajeong84@gmail.com

Abstract. The advancement of quantum computing poses a serious threat to widely-used public-key cryptographic systems, leaving organizations with the challenge of identifying quantum-vulnerable (QV) executables within their systems and migrating to post-quantum cryptography (PQC). The prior work QED achieves high detection precision through angr-based static call graph analysis (P3), but its processing time of a few to over ten minutes per dataset and RAM consumption of up to 5 GB pose practical limitations on exhaustive screening in large-scale environments. In this paper, we propose QED-Lite, a lightweight approximation tool that lightens the analysis stages that incur high computational overhead relative to their detection precision, replacing them with version-based PQC support determination of cryptographic libraries. QED-Lite introduces a newly built PQC risk classification database for 11 major cryptographic libraries and is implemented with a single dependency on pyelftools. Experimental results on the Network dataset show that QED-Lite achieves an execution time up to 855× faster (0.84 seconds) and reduces memory usage by 228× (22.9 MB) compared to QED, while maintaining a TPR of 100%. QED-Lite delivers practical value relative to its cost as a first-pass screening tool that is operationally viable in large-scale organizational environments.

Keywords: Post-Quantum Cryptography · Binary Analysis · ELF Static Analysis · PQC Migration · Quantum-Vulnerable Detection

1 Introduction

The "Harvest Now, Decrypt Later (HNDL)" attack scenario—in which currently unbreakable encrypted communications are collected in advance and decrypted once quantum computers become practical—has made the transition to post-quantum cryptography (PQC) increasingly urgent [1]. In response, the National

Institute of Standards and Technology (NIST) has recently completed its PQC standardization effort by officially adopting ML-KEM (FIPS 203), ML-DSA (FIPS 204), and SLH-DSA (FIPS 205) as formal standards [9]. With the completion of this standardization, the paradigm of information security is sharply shifting from designing new algorithms to migrating existing software systems across organizations to PQC. The first step for a successful PQC migration is to comprehensively identify all quantum-vulnerable (QV) cryptographic assets scattered throughout the system. However, in large-scale organizational environments where tens of thousands of third-party and legacy executables (binaries) without source code are intermixed, exhaustively surveying which files depend on QV cryptography constitutes a practical bottleneck requiring enormous time and resources.

To address this need, Rattanavipanon et al. proposed QED (Quantum-vulnerable Executable Detection), a static analysis-based binary vulnerability detection tool [11]. QED detects QV executables by sequentially performing three stages on ELF binaries: file-level dependency analysis (P1), API-level dependency analysis (P2), and angr framework-based static call graph analysis (P3). This design pursues a balance between efficiency and accuracy by rapidly narrowing down candidates in the early stages and performing precise analysis only in the final stage. However, from a practical operational perspective, QED has two critical limitations for its use as a large-scale screening tool. First, the in-depth call graph analysis causes excessive processing time and memory consumption per executable, making it unsuitable for exhaustive screening. Second, it has a structural blind spot in that it cannot trace indirect calls through function pointers, resulting in false negatives for some QV executables. In other words, despite consuming enormous resources, it cannot fully guarantee the prevention of missed threats, which is the core requirement of first-pass screening.

In this paper, we propose QED-Lite, taking these limitations into account. QED-Lite is a lightweight approximation tool that inherits P1 (file-level dependency analysis) from QED, while lightening the analysis stages that incur high computational overhead relative to their detection precision and directly determining PQC support of detected cryptographic libraries based on version information. Instead of tracing "which QV APIs are called," it rapidly outputs a risk classification based on "whether the linked cryptographic library is a version that supports PQC." Although this introduces a tradeoff of higher FPR, from the perspective of a first-pass screening tool, the priority is to miss no QV executables (maximizing TPR), and FPs can be handled through subsequent manual review by security analysts.

The contributions of this paper are as follows.

1. **Design and implementation of a lightweight quantum-vulnerable executable detection and migration planning support pipeline.** We lightened the analysis stages in QED that incur high computational overhead relative to their detection precision, resulting in an approximation pipeline that implements file-level dependency analysis via pyelftools-based static ELF parsing. By completely removing angr and its related dependencies, the

tool operates immediately with a single dependency (pyelftools) in any Linux environment supporting Python 3.8 or later.

2. **Construction of a PQC risk classification database for 11 cryptographic libraries.** We systematically compiled PQC support threshold versions based on official release notes for 11 major cryptographic libraries including OpenSSL, wolfSSL, and mbedTLS, and structured them into four tiers: `QUANTUM_VULNERABLE`, `HYBRID`, `PQC_READY`, and `NO_CRYPTO_DEP`. We also implemented a migration planning support feature that automatically suggests a stepwise upgrade path based on detection results.
3. **Empirical performance validation.** Measuring performance on the same datasets as QED, QED-Lite detects all QV executables in the evaluated datasets including curl, which QED P3 misses due to indirect calls through function pointers (Network TPR 100%), and achieves an execution time up to 855× faster (0.84 seconds) and reduces RAM usage by 228× (22.9 MB) compared to QED.

The full source code of QED-Lite proposed in this paper is publicly available at `https://github.com/HGKim442/QED-Lite`.

2 Related Work

2.1 PQC Support Status and Version Gap by Cryptographic Library

Cryptographic libraries currently widely used in Linux environments differ in the timing and manner of their adoption of the NIST PQC standards [9], suggesting the need for fine-grained risk determination at the version level.

OpenSSL, the de facto standard in general-purpose server environments, released version 3.5 in April 2025, natively incorporating ML-KEM, ML-DSA, and SLH-DSA [10]. Versions 3.0 through 3.4 can support PQC via the oqs-provider plugin, but whether the plugin is installed is difficult to detect through static analysis. Version 1.x provides no PQC support whatsoever. wolfSSL fully incorporated the NIST standards in version 5.8.0, released in April 2025 [12]. However, since PQC support depends on compile-time options, there is a structural limitation in that the activation status cannot be confirmed by version alone. mbedTLS is a lightweight library primarily targeting IoT and MCU environments, and does not support NIST PQC standard algorithms even in its latest version 4.0.0 [7].

Among others, GnuTLS has recently introduced some PQC features but has not yet reached the level of formal standardization, while NSS version 3.105 and above, and BoringSSL across all versions, formally support PQC. libsodium and liboqs have their own limitations: the former lacks formal NIST PQC support, and the latter is intended solely for research and prototype use.

As such, even the same cryptographic library can fall under different risk levels depending on the version, ranging from no PQC support, to transitional support, to formal support. Therefore, the mere fact of using a cryptographic library is insufficient to determine QV risk, and fine-grained determination at the version level is essential.

2.2 Binary-Based Cryptographic Detection

Research on identifying cryptographic implementations in binaries can be categorized into dynamic analysis, static analysis, and symbolic execution-combined approaches.

Dynamic analysis approaches detect cryptographic routines based on information obtained by actually executing the binary. Gröbert et al. proposed a method that collects execution traces using a dynamic binary instrumentation (DBI) framework and applies heuristics based on characteristics of cryptographic code—such as bitwise instruction ratios, loop structures, and input-output relationships—to identify various cryptographic primitives [3]. This approach provides high detection accuracy by directly observing execution paths, but raises safety concerns when analyzing malicious binaries and is difficult to automate in large-scale environments.

Static analysis approaches directly analyze binary structures without execution. Lestringant et al. proposed a static identification method for symmetric cryptographic algorithms that is robust against compiler options and source code transformations, using data flow graph (DFG) isomorphism [6]. While it has the advantage of producing no false positives, it is limited to identifying known algorithms and has difficulty handling code that contains conditional branches.

Approaches combining symbolic execution have also been proposed. Meijer et al. proposed a method that combines DFG isomorphism and symbolic execution to automatically identify and classify even unknown proprietary cryptographic primitives [8]. By defining the structural characteristics of cryptographic primitives using a domain-specific language (DSL) and resolving loop and conditional branch issues through symbolic execution, they overcame the limitations of existing DFG approaches.

These studies commonly aim to identify which cryptographic algorithms are implemented within a binary. In contrast, this study aims to determine whether the version of the cryptographic library linked to a binary supports PQC, which is a distinct problem from cryptographic implementation identification.

2.3 PQC Migration Support Tools and Related Work

Tools supporting PQC migration vary widely depending on the analysis target (source code/binary) and the operational approach (static analysis/runtime monitoring). They commonly focus on building organization-wide cryptographic asset inventories and migration management.

A representative example of source code-based tools is IBM Quantum Safe Explorer [4]. This tool statically analyzes source code to detect cryptography-related artifacts, automatically generates a Cryptography Bill of Materials (CBOM), and supports immediate remediation through IDE integration. It provides high precision in environments where source code access is available, and enables continuous cryptographic vulnerability monitoring through integration with CI/CD pipelines.

A tool combining binary and network analysis is Keyfactor's AgileSec Analytics [5]. This tool provides multi-layer inspection covering source code, binaries, and network traffic, and supports organization-wide cryptographic asset inventory construction and continuous monitoring.

These tools are designed under the premise of source code accessibility or runtime environment configuration, making them effective for organization-level PQC migration management. However, they are not directly optimized for scenarios that require rapidly screening large volumes of black-box ELF binaries offline without source code.

2.4 QED Pipeline Structure and Limitations

Among studies that detect QV executables using only ELF binaries without source code, QED (Quantum-vulnerable Executable Detection) [11] is the most directly relevant prior work. QED sequentially performs three stages—file-level dependency analysis (P1), API-level dependency analysis (P2), and angr framework-based static call graph analysis (P3)—where each stage takes the output of the previous stage as input, progressively narrowing the scope of analysis. This design pursues a balance between efficiency and accuracy by rapidly narrowing down candidates in the early stages and performing precise analysis only in the final stage.

However, there are two practical constraints in operating QED as a first-pass screening tool in large-scale environments. First, the resource overhead of P3. angr-based static call graph analysis incurs the largest resource overhead, requiring several gigabytes of memory and over ten minutes of analysis time in large-scale binary environments. Second, false negatives arising from the structural limitations of static analysis. Indirect calls through function pointers are not traced by CFGFast, which may result in executables that indirectly call QV APIs being missed.

This study is not intended to replace QED, but rather to serve as a lightweight approximation that redesigns the same problem for a different operational purpose. While QED aims for precise detection, QED-Lite targets rapid first-pass screening in large-scale environments.

Taken together, the following research gaps exist. Existing binary cryptographic detection research focuses on identifying cryptographic algorithm implementations, which differs in goal from QV library version determination. Existing PQC migration support tools presuppose source code access or runtime monitoring, making them not directly suited for black-box binary environments. QED is the closest prior work to ELF binary-based QV detection, but the resource cost of its precise analysis stage is a burden for large-scale first-pass screening. Therefore, a tool that rapidly performs first-pass screening of QV executables in large-scale environments using only ELF binaries without source code, based on cryptographic library version information, does not yet exist, and this study addresses this gap.

3 Implementation

3.1 QED Baseline Analysis

To determine the design direction of QED-Lite, we first measured the performance of QED empirically. QED consists of three stages—file-level dependency analysis (P1), API-level dependency analysis (P2), and angr framework-based static call graph analysis (P3)—where each stage takes the output of the previous stage as input and executes sequentially.

Empirical measurements confirmed that P3 accounts for the majority of the total execution time. Based on the Network dataset (13 ELFs), P1 took 1.016 seconds and P2 took 0.375 seconds, whereas P3 took 703.008 seconds, accounting for 97.9% of the total execution time (718 seconds). In the tpm2-tools dataset (86 ELFs) as well, P3 took 457.843 seconds, accounting for 97.4% of the total (470 seconds). Memory usage during P3 execution also reached a peak of 5,216 MB on the Network dataset and 3,476 MB on the tpm2-tools dataset.

Table 1. QED Per-Stage Execution Time Analysis

Dataset	P1	P2	P3	Total	P3 Ratio
Network (13 ELFs)	1.016s	0.375s	703.008s	718s	97.9%
tpm2-tools (86 ELFs)	1.745s	0.729s	457.843s	470s	97.4%

† Coreutils and UnixBench are excluded because no cryptographic library dependencies were detected in P1, so P2 and P3 were not executed; Synthetic is excluded because per-phase measurements are not separately output.

This indicates that operating QED as an exhaustive screening tool in large-scale organizational environments with thousands to tens of thousands of executables poses a significant practical burden.

Furthermore, despite its high precision, P3 has limitations in detection accuracy. curl indirectly calls QV APIs through function pointers, and angr's CFGFast cannot statically trace such indirect call paths. As a result, QED P3 classifies curl as a false negative. This demonstrates that even with enormous resource investment, the fundamental limitations of static call graph analysis prevent guaranteeing complete detection.

From the above analysis, we confirmed that eliminating P3 is the key design decision. P1 and P2 combined complete within a few seconds, and the cause of overall resource consumption lies virtually entirely with P3 alone.

3.2 Design Overview

QED-Lite is a lightweight approximation tool that inherits file-level dependency analysis from QED, while lightening the analysis stages that incur high computational overhead relative to their detection precision and directly determining PQC

support of detected cryptographic libraries based on version information. Instead of tracing "which QV APIs are called," it uses "whether the linked cryptographic library is a version that supports PQC" as the basis for determination.

Since ldd used in QED P1 operates by actually invoking the dynamic linker, errors may occur when the target binary is malicious or does not match the host architecture. QED-Lite replaces this with pyelftools-based static ELF parsing. pyelftools is safe as it directly parses ELF structures without executing the binary, is architecture-independent, and operates immediately with a single dependency (pyelftools) in any Linux environment supporting Python 3.8 or later without additional configuration.

QED-Lite is positioned not as a "replacement" for QED, but as a "lightweight approximation." Due to the removal of P2, binaries that are linked to cryptographic libraries but do not actually call QV APIs may be classified as false positives. This is an anticipated tradeoff by design, and a deliberate choice to prioritize detection without false negatives (maximizing TPR) as a first-pass screening tool.

3.3 Pipeline Structure

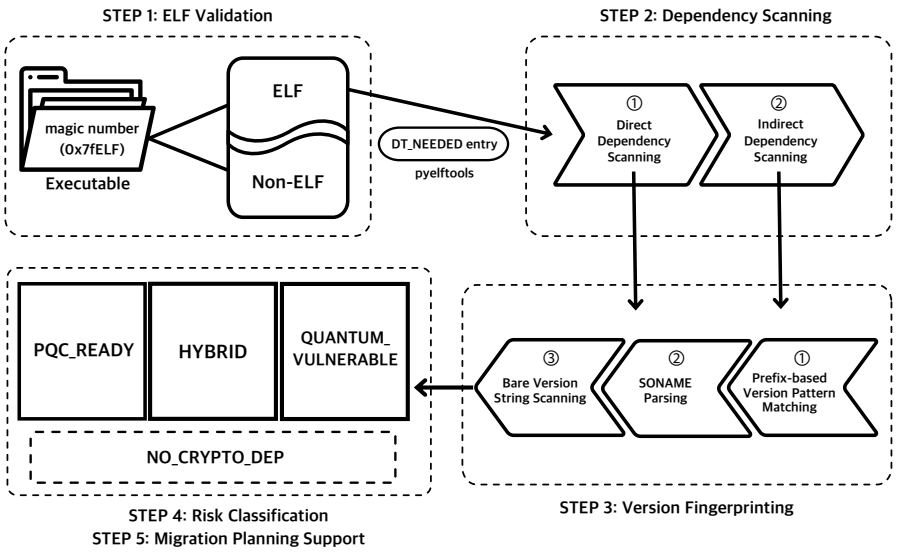

Fig. 1. QED-Lite Analysis Pipeline Structure

The analysis of QED-Lite consists of the following five stages.

STEP 1: ELF Validation. The magic number (0x7fELF) of the target file is verified. Non-ELF files are excluded from analysis.

STEP 2: Dependency Scanning. DT_NEEDED entries of the target ELF are extracted using pyelftools, and library names are resolved to full paths via the ldconfig -p output cached once at process startup. Cryptographic library pattern matching is performed in two stages. In the first stage, cryptographic libraries are searched among the direct dependencies of the target ELF, and returned immediately if found. If not found in the first stage, a second-stage scan is performed by additionally checking the DT_NEEDED entries of the direct dependency libraries. For example, nmap has only libssl in its direct dependencies, but by additionally checking the DT_NEEDED entries of libssl, the indirect dependency on libcrypto can be detected. If no cryptographic library is found through the second stage, the analysis terminates early with `NO_CRYPTO_DEP`.

STEP 3: Version Fingerprinting. The version of the detected cryptographic library is extracted and compared against the version DB to determine PQC support. Version extraction attempts three strategies in order. First, matching is attempted using library name prefix patterns (e.g., "wolfSSL X.Y.Z", "OpenSSL X.Y.Z") in the ELF `.comment` and `.rodata` sections. If this fails, SONAME-based extraction is attempted by parsing the version number from the actual path of the library file (e.g., `libcrypto.so.3` → `(3,0,0)`). Finally, for libraries whose SONAME encodes the ABI version (e.g., wolfSSL), the version string is directly searched in the `.rodata` section. If version extraction fails, a conservative fallback of `QUANTUM_VULNERABLE` is applied.

STEP 4: Risk Classification. The extracted version is compared against the version DB to determine the final classification (`QUANTUM_VULNERABLE` / `HYBRID` / `PQC_READY` / `NO_CRYPTO_DEP`) and output the result.

STEP 5: Migration Planning Support. Taking the output of STEP 4 (risk classification and detected library information) as input, it automatically generates immediate mitigation measures and a stepwise upgrade path of `QUANTUM_VULNERABLE` → `HYBRID` → `PQC_READY`.

3.4 PQC Support Determination Mapping Table

QED's `crypto_desc.py` defines only the list of QV APIs for each cryptographic library, and contains no version-based safe/vulnerable determination logic. For QED-Lite, `version_db.py` was newly constructed, which systematically compiles PQC support threshold versions based on official release notes for 11 cryptographic libraries: OpenSSL, wolfSSL, mbedTLS, BoringSSL, LibreSSL, libsodium, Botan, GnuTLS, NSS, liboqs, and libgcrypt. The determination criterion is whether NIST FIPS 203 (ML-KEM), FIPS 204 (ML-DSA), and FIPS 205 (SLH-DSA) have been officially adopted as formal standards [9].

Since BoringSSL does not use semver, it is classified as `PQC_READY` based on builds shipped with Chrome 131 or later, rather than a version range [2]. liboqs is a research and prototype-only library, and since the development team does not officially recommend its use in production, all versions are classified as `HYBRID` with a separate research flag (`is_research_grade`) noted. OpenSSL 3.0–3.4 can support PQC via the oqs-provider plugin, but since the plugin installation cannot be confirmed through static analysis, it is conservatively classified as

Table 2. PQC Risk Classification Mapping Table by Cryptographic Library

Library	QUANTUM VULNERABLE	HYBRID	PQC_READY
OpenSSL	< 3.0	3.0 – 3.4	≥ 3.5
wolfSSL	—	< 5.8	≥ 5.8
mbedTLS	All versions	—	—
BoringSSL	—	—	All versions[†]
LibreSSL	< 4.1	≥ 4.1	—
libsodium	All versions	—	—
Botan	—	2.x – 3.5	≥ 3.6
GnuTLS	$< 3.8.8$	$\geq 3.8.8$	—
NSS	< 3.97	3.97 – 3.104	≥ 3.105
liboqs	—	All versions[‡]	—
libgcrypt	< 1.11	≥ 1.11	—

[†] No semver; all builds from Chrome 131+ are PQC-ready.
[‡] Research/prototype use only; not recommended for production.

`HYBRID`. Since mbedTLS does not support NIST PQC standard algorithms in any version released to date, all versions are classified as `QUANTUM_VULNERABLE`. LibreSSL 4.1.0 and above have internally introduced ML-KEM, but since it is not exposed as a public API and cannot be directly used by applications, it is classified as `HYBRID`. GnuTLS 3.8.8 and above experimentally support ML-KEM in TLS 1.3, but since availability varies depending on distribution build settings, it is classified as `HYBRID`. libgcrypt 1.11.0 and above have added the Kyber KEM API, but since it is a pre-standard implementation rather than the final FIPS 203 form, it is classified as `HYBRID`. For wolfSSL versions prior to 5.8.0, since PQC support depends on compile-time options and build configurations, the vulnerability status cannot be confirmed, and thus it is classified as `HYBRID`. This is a structural limitation of static analysis, which is discussed in Section 5.2.

3.5 Risk Classification Definition

QED-Lite outputs analysis results in the following four classifications.

QUANTUM_VULNERABLE: The linked cryptographic library is a version that does not support PQC. It uses QV algorithms such as RSA, ECC, and DH, and migration to PQC is required.

HYBRID: The linked cryptographic library is a transitional version that supports both classical cryptography and PQC. PQC application is technically possible, but whether it is actually activated depends on configuration and build options.

PQC_READY: The linked cryptographic library is a version that formally supports PQC based on FIPS 203/204/205.

NO_CRYPTO_DEP: No cryptographic library dependency was detected within the analysis scope. This applies to cases where cryptography is not used, or where the library is connected via static linking or indirect linking of three or more hops.

3.6 Implementation Details

QED-Lite was implemented by removing unnecessary components from QED and adding new features. The entire codebase is written in Python, and the core dependency consists of a single package, pyelftools. We removed all heavy dependencies of QED, including angr, networkx, and matplotlib.

The key component files are as follows. `qed_lite.py` is the main entry point, modified from QED's `qed.py` by removing the P2 and P3 call sections and adding version fingerprinting and result output logic. `FileDependencyAnalysis.py` is responsible for the P1 analysis logic, replacing ldd-based dependency scanning with pyelftools `DT_NEEDED` parsing and adding two-stage scanning and version extraction logic. `version_db.py` is a newly written file serving as the PQC risk classification database for 11 cryptographic libraries. `migration_guide.py` is responsible for the migration planning support of STEP 5.

4 Evaluation

4.1 Experimental Setup and Datasets

All experiments were conducted on a system with an Intel Core i7-10875H (8 cores, 2.30 GHz), 16 GB RAM, and Ubuntu 24.04 (x86_64). Execution time and memory usage were recorded as the mean of three runs using `/usr/bin/time -v`. This evaluation focuses on detection accuracy and performance; the migration planning support output of STEP 5 is addressed in the pipeline description of Section 3.3.

The datasets were constructed to match those of the prior work [11], comprising one synthetic dataset and four real-world datasets for a total of 266 ELF binaries (Table 3). The Network dataset, which contains a mix of cryptographic and non-cryptographic programs, served as the reference dataset for detection accuracy, while tpm2-tools was used mainly for performance measurement, as cryptographic API polysemy and runtime-dependent behavior render Ground Truth determination infeasible through static binary analysis alone. Both tools were evaluated on identical hardware and datasets; QED ran under Python 3.10 with angr 9.2.90, and QED-Lite under Python 3.12.3 with pyelftools 0.32.

4.2 Execution Time and Memory Usage Comparison

On the Network and tpm2-tools datasets, which contain cryptographic library dependencies, QED-Lite achieved speedups of 855× and 326×, respectively. Memory usage was reduced by 228× on the Network dataset (from 5,216 MB to

Table 3. Experimental Dataset Composition

Dataset	ELF Count	Type	Ground Truth
Synthetic	40	Synthetic (20 direct / 20 indirect)	Fully manual
Coreutils	109	Real-world (non-crypto)	All Non-QV
UnixBench	18	Real-world (non-crypto)	All Non-QV
Network	13	Real-world (crypto included)	Manually verified
tpm2-tools	86	Real-world (crypto included)	Undeterminable

Table 4. QED vs. QED-Lite: Execution Time and Memory Usage Comparison (Average of Three Runs)

Dataset	ELF Count	QED Time	QED-Lite Time	Speedup	QED RAM	QED-Lite RAM
Synthetic	40	17.33s	0.54s	32x	324 MB	21.3 MB
Coreutils	109	2.09s	0.60s	3.5x	153 MB	20.8 MB
UnixBench	18	1.22s	0.18s	6.8x	153 MB	20.2 MB
Network	13	718s	0.84s	855x	5,216 MB	22.9 MB
tpm2-tools	86	470s	1.44s	326x	3,476 MB	21.5 MB
Real-World Total*	226	1,136s	2.55s	445x	8,597 MB	23.7 MB

* Results of a single unified process run across all four datasets; discrepancies from the per-dataset sum are due to deduplication of shared library CFGs.

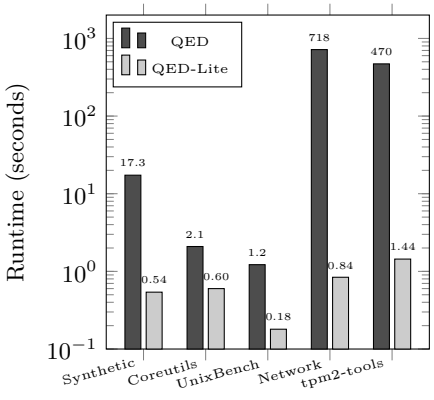

Fig. 2. Execution Time Comparison

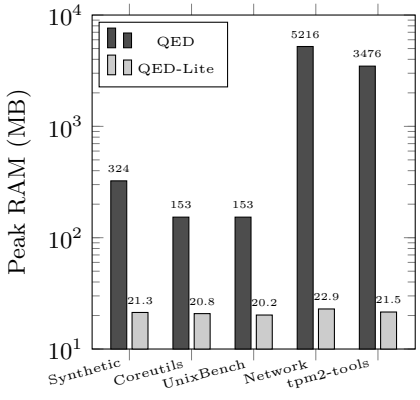

Fig. 3. Memory Usage Comparison

22.9 MB), by 162× on tpm2-tools, and by 363× across all real-world datasets combined. As confirmed in Section 3.1, this is because P3 accounts for over 97% of the total execution time (97.9% on Network, 97.4% on tpm2-tools), making the elimination of P3 the primary driver of the performance improvement.

In contrast, speedups are relatively modest on datasets without cryptographic library dependencies, such as Coreutils and UnixBench. QED is inherently fast on these datasets because P1 eliminates all such files before P3 is reached, while QED-Lite exhibits a smaller relative gap in extremely short runs due to the overhead of initializing the `ldconfig` cache.

4.3 Detection Accuracy Comparison

Accuracy measurement was performed on the Network and Synthetic datasets, for which Ground Truth was established.

QED-Lite achieved a TPR of 100% on the Network dataset. curl, which QED had classified as an FN, was detected as a TP by QED-Lite, because curl is indirectly linked to `libcrypto.so.3` via `libcurl`, enabling its detection through version-based determination (HYBRID). QED P1 and QED-Lite achieved identical accuracy on the Network dataset, with a TPR of 100% and a TNR of 66.7%. This outcome reflects QED-Lite inheriting the conservative detection tendency of QED P1 (Table 5), while further resolving the false negative on curl through version-based determination. In contrast, QED eliminated all FPs with a TNR of 100%, but missed curl, resulting in a TPR of only 85.7%.

Table 5. Per-Application Determination Results on the Network Dataset

App	Ground Truth	QED (P1)	QED (P3)	QED-Lite
ssh	QV	TP	TP	TP (QV)
sshd	QV	TP	TP	TP (QV)
nmap	QV	TP	TP	TP (QV)
wget	QV	TP	TP	TP (QV)
dig	QV	TP	TP	TP (QV)
nslookup	QV	TP	TP	TP (QV)
curl	QV	TP	FN	TP (HYBRID)
scp	Non-QV	FP	TN	FP (QV)
sftp	Non-QV	FP	TN	FP (QV)
ping	Non-QV	TN	TN	TN (NCD)
netcat	Non-QV	TN	TN	TN (NCD)
telnet	Non-QV	TN	TN	TN (NCD)
tracepath	Non-QV	TN	TN	TN (NCD)

QV: QUANTUM_VULNERABLE, NCD: NO_CRYPTO_DEP

On the Synthetic dataset, QED-Lite achieved a TPR of 100% (24/24) while its TNR was 0% (0/16). This is because programs using only non-QV algorithms

such as AES and SHA-512 are also flagged if they are linked to a cryptographic library. This stems from the structural characteristic of relying solely on file-level dependencies without P2.

The tpm2-tools dataset was excluded from accuracy measurement as Ground Truth determination is infeasible. However, QED-Lite detected all 86 binaries as `QUANTUM_VULNERABLE`, because all tpm2 binaries are either directly linked to `libcrypto.so.1.1` or indirectly linked to `libcrypto.so.3` via `libtss2-esys`. QED detected only 13 binaries for which actual QV API call paths were confirmed, whereas QED-Lite detected all 86 based on file-level dependencies. This is a structural characteristic arising from the removal of P2, and actual QV status warrants follow-up review using a precise analysis tool such as QED.

In summary, QED-Lite achieved a TPR of 100% on both datasets. QED-Lite detected curl, which QED missed on the Network dataset due to indirect calls through function pointers, and all 6 indirect cases that QED missed on the Synthetic dataset due to the limitations of static analysis. However, the TNR dropped to 0% on the Synthetic dataset due to the removal of P2. This demonstrates that QED-Lite is a conservative first-pass screening tool that prioritizes minimizing FNs, a characteristic clearly reflected in Table 6 and Table 7.

Table 6. Accuracy Comparison on the Network Dataset

	QED	QED-Lite
TP (QV Detected)	6/7	7/7
FN (QV Missed)	1	0
FP (Non-QV Misclassified)	0/6	2/6
TN (Non-QV Correct)	6/6	4/6
TPR	85.7%	100%
TNR	100%	66.7%

Table 7. Accuracy Comparison on the Synthetic Dataset

	QED		QED-Lite	
	Direct	Indirect	Direct	Indirect
TP (QV Detected)	12/12	6/12	12/12	12/12
FN (QV Missed)	0	6	0	0
FP (Non-QV Misclassified)	0/8	0/8	8/8	8/8
TN (Non-QV Correct)	8/8	8/8	0/8	0/8
TPR	100%	50%	100%	100%
TNR	100%	100%	0%	0%

4.4 Analysis and Discussion

FP Cause Analysis. scp and sftp are dynamically linked to `libcrypto` but use only non-QV APIs such as `OPENSSL_init_crypto` and `RAND_bytes`. QED P2 identifies and excludes these through API-level analysis, whereas QED-Lite classifies them as FPs because it relies solely on file-level dependencies without P2. This is a structural tradeoff anticipated in the lightweight approximation design.

Tradeoff Discussion. QED-Lite exhibits a conservative detection tendency with high TPR and high FPR. From the perspective of a first-pass screening tool, the priority is to miss no QV executables (maximizing TPR), and FPs can be handled through subsequent manual review by security analysts. Indeed, the prior work [11] also provides an option to tolerate FPs and eliminate FNs in conservative mode, aligning QED-Lite's detection tendency with the existing design philosophy.

Limitations. Statically linked cryptographic libraries are not detected, and indirect linking of three or more hops falls outside the detection scope. For libraries such as wolfSSL whose PQC support depends on compile-time options, activation status cannot be confirmed by version alone. Additionally, since a conservative fallback of `QUANTUM_VULNERABLE` is applied when version extraction fails, even the latest library version may be classified as vulnerable if version information is unavailable.

5 Conclusion

5.1 Summary

In this paper, we proposed QED-Lite, a lightweight approximation tool to overcome the practical limitations of QED. QED-Lite was designed to lighten the analysis stages that incur high computational overhead relative to their detection precision, and to directly determine PQC support based on the version information of detected cryptographic libraries. We also constructed a PQC risk classification database for 11 major cryptographic libraries based on official release notes, and implemented a migration planning support feature that automatically suggests a stepwise upgrade path based on detection results.

Experimental results demonstrated that QED-Lite maintained a TPR of 100% on the Network dataset while achieving an execution time up to 855× faster (0.84 seconds) and reducing memory usage by 228× (22.9 MB) compared to QED. This demonstrates that QED-Lite constitutes a practical choice relative to its cost as a first-pass screening tool that is operationally viable in large-scale organizational environments.

5.2 Limitations

QED-Lite has several limitations arising from its lightweight approximation design. Due to the removal of P2, binaries that are linked to cryptographic

libraries but do not actually call QV APIs may be classified as FPs, and since only dynamic linking is targeted, statically linked cryptographic libraries fall outside the detection scope. Additionally, since indirect dependency scanning is limited to two hops, indirect linking of three or more hops cannot be detected, and for libraries such as wolfSSL whose PQC support depends on compile-time options, activation status cannot be confirmed by version alone.

5.3 Future Work

The following future research directions are proposed to address the limitations of QED-Lite.

First, static linking detection. Since QED-Lite currently targets only dynamic linking (`DT_NEEDED`), statically compiled cryptographic libraries fall outside the detection scope. The detection scope can be extended by adding static linking detection techniques based on the ELF `.rodata` section and symbol table.

Second, extending indirect dependency scanning depth. The fixed two-hop scanning depth can be generalized into a configurable parameter, or a methodology for determining the optimal scanning depth based on real-world deployment data can be explored.

Third, building a progressive PQC migration framework through integration of lightweight screening with DevOps/CI-CD pipelines. Since even precise detection tools cannot guarantee complete detection due to the structural limitations of static analysis, investing additional resources solely to improve detection accuracy remains inherently limited. A more practical approach is to build an operational framework that integrates QED-Lite into CI/CD pipelines to continuously perform first-pass screening and progressively track and supplement undetected binaries. By doing so, a practical PQC migration framework can be proposed that compensates for the limitations of incomplete detection via continuous monitoring.

Fourth, building an automated `version_db` update system in response to PQC standard updates. Since `version_db.py` is currently managed manually, it is difficult to immediately reflect NIST standard updates or new releases of cryptographic libraries. Building a system that automatically collects and parses official release notes and changelogs to update `version_db` would greatly improve the long-term maintainability of the tool.

References

1. Ahmed, N., Zhang, L., Gangopadhyay, A.: A survey of post-quantum cryptography support in cryptographic libraries. In: 2025 IEEE International Conference on Quantum Computing and Engineering (QCE). vol. 01, pp. 906–917 (2025). https://doi.org/10.1109/QCE65121.2025.00102
2. Google Security Blog: A new path for Kyber on the web. https://security.googleblog.com/2024/09/a-new-path-for-kyber-on-web.html (Sep 2024), accessed: 2026-03-25

3. Gröbert, F., Willems, C., Holz, T.: Automated identification of cryptographic primitives in binary programs. In: Proceedings of the 14th International Symposium on Recent Advances in Intrusion Detection (RAID). Lecture Notes in Computer Science, vol. 6961, pp. 41–60. Springer (2011)
4. IBM: Empowering CIOs to accelerate crypto-agility with IBM quantum safe explorer. https://www.ibm.com/new/product-blog/empowering-cios-to-accelerate-crypto-agility-with-ibm-quantum-safe-explorer (Dec 2025), accessed: 2026-03-20
5. Keyfactor: Discover & inventory cryptography with AgileSec™ analytics. https://www.infosecglobal.com/products/agilesec-analytics (2025), accessed: 2026-03-20
6. Lestringant, P., Guihéry, F., Fouque, P.A.: Automated identification of cryptographic primitives in binary code with data flow graph isomorphism. In: Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security (ASIACCS). pp. 203–214. ACM (2015). https://doi.org/10.1145/2714576.2714639
7. Mbed TLS Contributors: Release mbed tls 4.0.0. https://github.com/Mbed-TLS/mbedtls/releases/tag/mbedtls-4.0.0 (Oct 2024), accessed: 2026-03-25
8. Meijer, C., Moonsamy, V., Wetzels, J.: Where's crypto?: Automated identification and classification of proprietary cryptographic primitives in binary code. In: Proceedings of the 30th USENIX Security Symposium. pp. 1919–1936. USENIX Association (2021)
9. National Institute of Standards and Technology: NIST releases first 3 finalized post-quantum encryption standards. https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards (Aug 2024), accessed: 2026-03-20
10. OpenSSL Library: OpenSSL 3.5 final release. https://openssl-library.org/post/2025-04-08-openssl-35-final-release/ (Apr 2025), accessed: 2026-03-20
11. Rattanavipanon, N., Suaboot, J., Werapun, W.: A toolchain for assisting migration of software executables towards post-quantum cryptography. IEEE Access **13**, 4368–4380 (2025). https://doi.org/10.1109/ACCESS.2024.3524309
12. wolfSSL Inc.: wolfssl 5.8.0 released. https://www.wolfssl.com/wolfssl-5-8-0-released/ (Apr 2025), accessed: 2026-03-20