ML-QED-Lite: A Lightweight Machine Learning-Based Tool for Supporting Post-Quantum Cryptography Migration in Executable Binaries

Seung-Won Lee[1][0009−0009−9583−1387] and Hwa-Jeong Seo[1][0000−0003−0069−9061]

Department of Information Computer Engineering,
Hansung University, Seoul, 02876 Korea
dkajdfhd1@gmail.com, hwajeong84@gmail.com

Abstract. To initiate migration to post-quantum cryptography (PQC), it is necessary to identify whether deployed software uses quantum-vulnerable (QV) public-key cryptographic schemes such as RSA, ECDSA, and Diffie–Hellman (DH). However, many ELF executables are distributed without source code, making it necessary to directly screen executable binaries for QV candidates. A prior tool, Quantum-vulnerable Executable Detection (QED), provides high precision but incurs substantial analysis cost, whereas its lightweight variant, QED-Lite, is faster but produces more false positives (FPs).

This paper proposes ML-QED-Lite, a machine learning-based approach designed to retain the efficiency of QED-Lite while reducing FPs. Unlike a post-filter that merely reclassifies candidates selected by QED-Lite, ML-QED-Lite directly takes all ELF executables in a target directory as input. For each file, it extracts function symbols, library dependencies, and binary-level attributes, and then uses a trained classification model to determine whether the file is a candidate for PQC migration.

The evaluation results show that ML-QED-Lite detects the same five positive executables as QED-Lite on the network dataset while reducing FPs from two to zero. On the synthetic dataset, ML-QED-Lite retains the same six positive executables while reducing FPs from four to zero. These results indicate that ML-QED-Lite preserves the true positives (TPs) identified by QED-Lite while reducing FPs, thereby improving the practicality of lightweight screening for PQC migration.

Keywords: Post-Quantum Cryptography · PQC Migration · ELF Binary Analysis · Machine Learning · Lightweight Screening

1 Introduction

If cryptographically relevant quantum computers become available, widely deployed public-key cryptographic schemes such as RSA and ECDSA will no longer provide adequate security. Shor's algorithm showed that integer factorization and discrete logarithm problems can be solved efficiently

on a quantum computer [14]. In response, NIST has standardized post-quantum cryptography (PQC) algorithms such as ML-KEM, ML-DSA, and SLH-DSA [11]. Consequently, existing public-key cryptographic systems must be migrated to PQC algorithms, a process commonly referred to as PQC migration. However, PQC migration is not merely a matter of selecting replacement algorithms. It is also a software transition problem that requires identifying where and how cryptography is used in deployed systems and prioritizing migration targets.

Previous studies have discussed migration procedures, stakeholder roles, and the need for dependency analysis to address this transition problem [10,4]. The first step is to identify which cryptographic algorithms are used and where they are used. The second step is to assess risk and prioritize migration targets. The third step is to establish an architecture that enables cryptographic algorithms to be replaced efficiently. Finally, before full deployment, the migration process should be validated through prototyping and incremental replacement [5]. In practice, however, deployed systems often contain executables for which source code is unavailable, or only legacy binaries remain. In such cases, source-code-level analysis alone is insufficient for locating cryptographic usage. Therefore, it is necessary to analyze deployed Executable and Linkable Format (ELF) binaries directly.

Quantum-vulnerable Executable Detection (QED) is a representative binary-level tool that directly addresses this problem in the context of PQC migration [13]. QED identifies executables that may use quantum-vulnerable (QV) public-key cryptography through file-dependency analysis, API analysis, and static trace analysis. This design provides high precision but incurs substantial analysis cost. The QED-Lite baseline considered in this study removes QED's static trace analysis and instead focuses on dependency information and library-version fingerprinting to reduce candidate sets quickly [7]. However, QED-Lite does not perform the same level of semantic analysis as the original QED. As a result, false positives (FPs) may arise during lightweight screening.

The goal of this study is not to claim that QED-Lite replaces QED. Rather, this study investigates whether machine learning can reduce FPs while preserving the fast candidate-detection capability of QED-Lite. To this end, we propose ML-QED-Lite, which directly takes ELF binaries as input. ML-QED-Lite is not a post-filter that merely re-examines candidates already selected by QED-Lite. Instead, it traverses all ELF executables in a given directory and extracts function symbols, library dependencies, and binary-level features from each file. The extracted features are then fed into a trained machine learning model, which directly classifies each executable as either a candidate for PQC migration or a non-candidate. In this way, ML-QED-Lite extends QED-Lite by improving the precision of lightweight screening.

The contributions of this paper are as follows.

1. We propose ML-QED-Lite, a machine learning-based extension designed to reduce FPs while preserving the lightweight nature of QED-Lite.

2. We design ML-QED-Lite as a direct binary-level classifier, rather than as a post-filter that reclassifies only the candidates selected by QED-Lite.
3. We quantitatively compare QED, QED-Lite, and ML-QED-Lite under the same experimental environment and analyze their differences in detection performance and analysis cost.

The remainder of this paper is organized as follows. Section 2 introduces QV cryptography, the key terminology used in this study, and the baseline tools QED and QED-Lite. Section 3 reviews prior work on PQC migration, binary-level cryptographic analysis, and evaluation bias in machine learning, and positions the problem addressed in this study. Section 4 describes the overall architecture of ML-QED-Lite, including feature extraction, model training, and threshold selection. Section 5 compares QED, QED-Lite, and ML-QED-Lite under the same experimental environment and analyzes the experimental results. Finally, Section 6 concludes the paper and discusses directions for future work.

2 Background

In this study, quantum-vulnerable (QV) cryptography refers to public-key cryptographic schemes that may become insecure in the presence of sufficiently powerful quantum computers. Public-key schemes such as RSA, DSA, ECDSA, DH, and ECDH are treated as QV schemes. In contrast, symmetric-key primitives and hash functions such as AES and the SHA family are outside the scope of QV detection in this study. We classify executables that may use QV cryptography as candidates for PQC migration that require further inspection.

As summarized in Table 1, a true positive (TP) is an actual QV file that is correctly detected as a PQC migration candidate. A false positive (FP) is an actual non-QV file that is incorrectly detected as a candidate. A true negative (TN) is an actual non-QV file that is correctly excluded, whereas a false negative (FN) is an actual QV file that is missed.

The true positive rate (TPR) indicates the proportion of actual QV files detected by a tool. The true negative rate (TNR) indicates the proportion of actual non-QV files correctly excluded by a tool. The false positive rate (FPR) indicates the proportion of actual non-QV files incorrectly classified as PQC migration candidates. A large number of FPs increases the number of files that must be manually inspected. Therefore, this study focuses on reducing FPs while preserving TPs.

2.1 QED

QED [13] is a tool designed to detect PQC migration candidates in ELF executables written in C/C++ on Linux. It exploits a common implementation pattern in which software does not directly implement cryptographic functionality, but instead dynamically links against external cryptographic libraries and invokes their APIs.

Table 1. Key terminology and evaluation metrics used in this study

Term	Meaning
TP	An actual QV file correctly detected as a candidate
FP	An actual non-QV file incorrectly detected as a candidate
TN	An actual non-QV file correctly excluded as not being a candidate
FN	An actual QV file that is missed
TPR	$TP/(TP+FN)$
TNR	$TN/(TN+FP)$
FPR	$FP/(FP+TN)$

QED consists of three main analysis stages. First, it performs file-level dependency analysis to determine which cryptographic libraries an executable depends on and quickly narrows down possible QV candidates. Second, it performs API analysis to check whether an executable that depends on a cryptographic library actually invokes QV-related APIs. Third, it performs static trace analysis to precisely verify whether the corresponding QV API calls are reachable from the main function in the execution flow. The original QED evaluation was conducted using both a synthetic dataset and a real-world dataset.

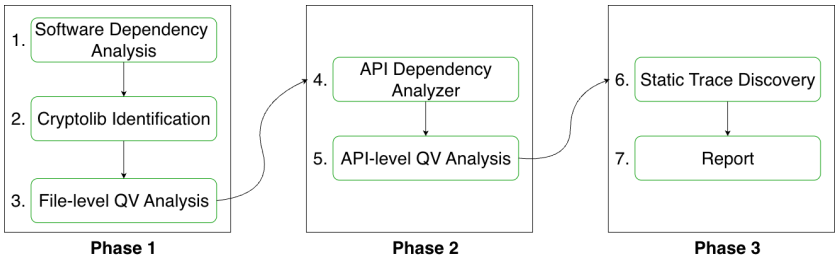

Fig. 1. QED pipeline

In the real-world dataset experiment, QED analyzes 226 files in approximately 15 min, corresponding to less than 4 s per file on average. However, its peak memory usage reaches approximately 3–5 GB, which is not negligible for lightweight screening.

2.2 QED-Lite

QED-Lite [7] is a fast scanning tool proposed to address the long analysis time, high memory usage, and limitations in function-pointer tracking observed in QED. QED-Lite removes the static trace analysis stage of QED and instead relies primarily on dependency information and library-version fingerprinting to quickly reduce the set of candidates.
The QED-Lite pipeline consists of five stages. First, it verifies whether the input file is a Linux executable and immediately excludes non-Linux

executables as an initial filtering step. Second, it obtains the list of linked libraries and checks whether the executable uses cryptographic libraries. Third, it extracts the versions of the detected cryptographic libraries and applies version-based filtering. If version extraction fails, the executable is conservatively classified as a QV candidate. Fourth, QED-Lite assigns a risk level to the file. Fifth, based on the assigned risk level, it provides guidelines for user action.

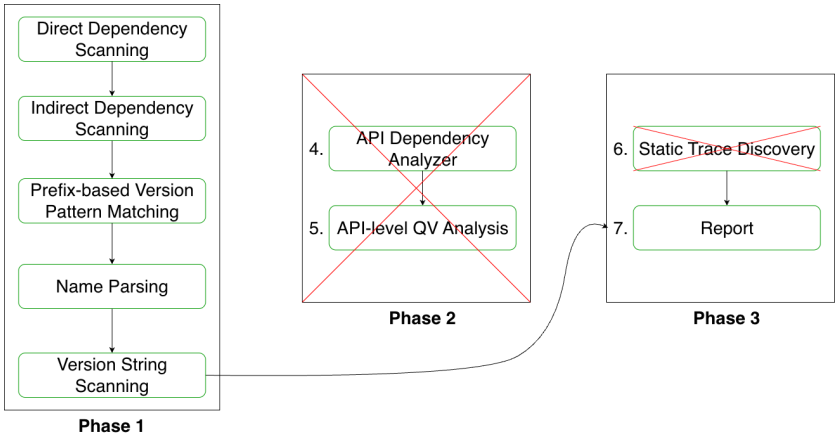

Fig. 2. QED-Lite pipeline

This approach improves efficiency over QED by factors of approximately 855 and 228 in terms of runtime and memory usage, respectively. However, it does not perform the same level of precise analysis as the original QED. Therefore, QED-Lite should be interpreted not as a replacement for QED, but as a lightweight tool for rapidly screening candidates that require further inspection during PQC migration.

3 Related Work

This section reviews prior work related to this study in three categories. The first category concerns studies on how PQC migration should be prepared and conducted. The second category concerns studies that identify cryptographic usage in binaries when source code is unavailable. The third category concerns the application of machine learning to binary analysis and the evaluation issues that must be considered when using machine learning models.

3.1 PQC Migration Procedures and Supporting Studies

PQC migration is the process of replacing existing public-key cryptographic schemes with PQC algorithms. It should not be regarded simply as replacing RSA with a PQC algorithm. One important reason is the need to consider cryptographic agility. Cryptographic agility refers to

the ability of a system to replace cryptographic algorithms easily when standards or security requirements change in the future [8]. PQC migration can be organized into four stages: diagnosis, planning, execution, and maintenance [10].

- **Diagnosis:** identifying which cryptographic algorithms are used and where they are used in the current system.
- **Planning**: determining which components should be migrated first based on the diagnosis results.
- **Execution**: replacing cryptographic algorithms, libraries, or protocols in practice.
- **Maintenance**: continuously managing the system after migration.

ML-QED-Lite, proposed in this study, belongs to the diagnosis stage. It aims to screen PQC migration candidates more efficiently and practically. Several studies have also proposed frameworks to support PQC migration in enterprise environments [4]. Such work connects data and assets and analyzes, using a graph-based representation, which cryptographic systems specific data assets depend on. Experimental results show that dependency analysis can help determine migration priorities. However, these approaches assume that an organization already maintains a certain level of asset inventory, and they do not directly address how to generate candidates by analyzing ELF binaries themselves.
In summary, existing PQC migration studies explain the importance of migration procedures, cryptographic inventories, dependency analysis, and cryptographic replaceability. However, they do not sufficiently address the problem of directly analyzing ELF executables without source code to rapidly identify QV candidates. This study focuses on that problem.

3.2 Binary-Based Cryptographic Analysis

Binary-based cryptographic analysis aims to identify traces of cryptographic usage directly from compiled executables when source code is unavailable. In real operational environments, source code may be missing, or only legacy executables may remain. Therefore, binary analysis is necessary in the diagnosis stage of PQC migration.
Prior studies have investigated methods for detecting cryptographic algorithms or cryptographic functions in binaries. Some approaches execute a program, collect execution-flow and memory-access information, and use characteristic patterns commonly observed in cryptographic code to identify cryptographic algorithms [3]. Other work represents data flow within a binary as a graph and compares it with graphs of known cryptographic algorithms [9]. Xu et al. [15] proposed CryptoHunt to identify cryptographic functions even in obfuscated binaries.

3.3 Machine Learning for Binary Analysis and Evaluation

Machine learning has also been applied to binary analysis. Arakelyan et al. [1] proposed Bin2vec, which converts binary executables into vectors

that can be used by machine learning models. Their approach learns structural and computational information from binaries and applies it to algorithm classification and vulnerability detection. This demonstrates that binary executables can be classified using machine learning. However, Bin2vec targets general binary security problems and does not directly address PQC migration candidate screening or FP control.

Although machine learning can be applied to binary analysis, the reliability of its results depends heavily on how the model is evaluated. In the security domain, poorly designed evaluation procedures can lead to overestimated performance. Kaufman et al. [6] explained that performance can be overestimated when a model is exposed during training to information that would not be available at prediction time. Pendlebury et al. [12] showed that, in security-oriented machine learning evaluation, randomly splitting data can place overly similar samples in both training and test sets, resulting in inflated performance estimates.

The results of a machine learning model also depend on how the decision threshold is selected. Lowering the threshold may detect more candidate files, but it can also increase FPs. Conversely, raising the threshold may reduce FPs, but it can also miss some QV candidates. Fawcett [2] summarized the importance of threshold selection in the evaluation of classification models. Therefore, after comparing candidate models, the threshold should be selected using separate validation data so that the FP rate does not exceed a predefined level.

3.4 Summary of Limitations in Prior Work

Table 2 summarizes how the limitations of prior work are connected to the design of this study.

Table 2. Comparison of key considerations between prior work and this study

Study	PQC Diagnosis	ELF Analysis	Lightweight Screening	FP Control
PQC migration process [8,10]	○	×	×	×
PQC migration framework [4]	○	×	×	×
QED [13]	○	○	×	○
QED-Lite [7]	○	○	○	△
Binary crypto analysis [3,9,15]	×	○	△	×
Binary ML research [1]	×	○	△	×
This study	○	○	○	○

Legend: ○ = included, △ = partial, × = not addressed

4 Implementation

This section describes the implementation of ML-QED-Lite. QED-Lite can rapidly reduce the set of candidates by relying on library dependencies and version information. However, the fact that an executable depends on a cryptographic library alone is not sufficient to determine whether it is an actual QV candidate. To address this limitation, ML-QED-Lite directly takes ELF executables as input, extracts symbols, library dependencies, and binary properties, and uses a trained classification model to determine whether each executable is a candidate for PQC migration.

The design of ML-QED-Lite inherits selected stages from QED-Lite while replacing only the decision mechanism. QED-Lite determines candidates through ELF filtering, library-dependency checking, and library-version fingerprinting. Among these stages, the version-based decision step is a major source of FPs. ML-QED-Lite keeps ELF filtering and library-dependency extraction identical to QED-Lite, but replaces the version-based decision step with a trained classifier. Instead of relying only on version information, ML-QED-Lite uses function symbols and binary properties that can be obtained through the same static-analysis process as additional evidence for candidate classification.

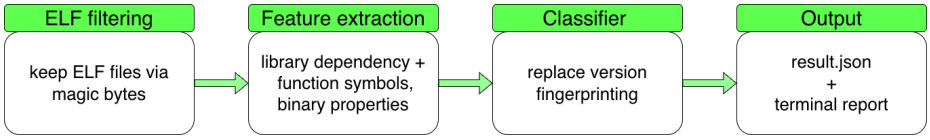

Fig. 3. ML-QED-Lite pipeline

The implementation consists of four main stages: ELF filtering, symbol and library extraction, feature construction, and model inference. This pipeline preserves the fast screening workflow of QED-Lite while reducing FPs that may arise when candidate decisions are made using only library-dependency information.

4.1 Feature Extraction

The overall structure of ML-QED-Lite is shown in Fig. 3. ML-QED-Lite first traverses an input directory and identifies ELF executables to be analyzed. It then uses `readelf` to extract symbols, library dependencies, and binary properties from each executable. Here, symbols refer to function names remaining in the executable. Library dependencies refer to the external shared libraries required at runtime. Binary properties refer to structural and security-related attributes of the executable, rather than function or library names. The extracted information is converted into features that can be used by a machine learning model. Finally, the classifier determines whether each executable is a PQC migration candidate or a non-candidate.

Feature extraction is implemented as follows. First, in the ELF filtering stage, ML-QED-Lite checks the first four bytes of each file to determine whether it is an ELF executable. Files that do not follow the ELF format are excluded from both feature extraction and candidate classification.
Second, in the symbol extraction stage, ML-QED-Lite uses `readelf --dyn-syms --wide` and `readelf -Ws`. The former is used to extract imported function symbols from external libraries, whereas the latter is used to extract the full set of function symbols. During extraction, only function names are used, and version suffixes appended to symbol names are removed. This normalization prevents the same function from being treated as different symbols solely because of library-version differences. Function names related to RSA, ECDSA, DH, and other public-key cryptographic operations are then used as indicators of possible QV API usage.
Third, in the library-dependency extraction stage, ML-QED-Lite uses `readelf -d` to obtain the list of shared libraries required by each executable. It checks whether the extracted list contains cryptographic libraries such as `libcrypto.so`, `libssl.so`, `libmbedtls.so`, and `libwolfssl.so`, and stores this information as library-dependency features.
Finally, ML-QED-Lite also uses binary-level properties as features. These include file size, ELF file type, whether symbol information has been stripped, whether the binary is statically linked, and security-related properties such as position-independent executable (PIE), non-executable stack (NX), and relocation read-only (RELRO).
These three feature categories provide complementary information. Symbols indicate which cryptographic APIs an executable may use. Library dependencies indicate which cryptographic libraries the executable is linked against. Binary properties provide additional evidence when symbol information is insufficient or library information alone is ambiguous. Therefore, rather than relying on a single source of evidence, ML-QED-Lite combines multiple static features to reduce FPs that may occur in QED-Lite.
In summary, ML-QED-Lite aggregates symbol names, QV- and non-QV-related symbol counts, library dependencies, and binary properties extracted from each executable. The resulting feature vector is then provided to a machine learning model to classify whether the executable is a candidate for PQC migration.

4.2 Training Data Construction

The training data for ML-QED-Lite were constructed from a collection of ELF executables. First, imported symbols were extracted from each ELF file, and files containing API symbols related to QV cryptography were assigned candidate labels. RSA, ECDSA, DH, and DSA were treated as QV public-key cryptographic schemes. In addition, OpenSSL EVP public-key APIs, TLS-connection-related symbols, and X.509-certificate-related symbols were also used for candidate labeling because they may indicate the use of public-key cryptography. In contrast, files that contained only non-QV cryptographic symbols, such as AES, SHA, HMAC, and RAND, without QV-related symbols were labeled as non-candidates.

Table 3. Features extracted by ML-QED-Lite

Feature group	Method	Included items
Symbol	`readelf --dyn-syms --wide`	RSA, ECDSA/EC, DH, DSA, EVP_PKEY, SSL, X.509 symbols
	`readelf -Ws`	AES, SHA, HMAC, RAND symbols
Library dependency	`readelf -d`	`libcrypto.so`, `libssl.so`, `libmbedtls.so`, `libmbedcrypto.so`, `libmbedx509.so`, `libwolfssl.so`, `libgcrypt.so`, `libbotan`, `libsodium.so`
Binary property	`readelf -h`	file size, `.text` section size, ELF type,
	`readelf -W -l`	PIE, NX, RELRO, stripped, static
	`readelf -S --wide`	linking, symbol count, needed-library count

Files for which neither criterion was clear were excluded from the training data.

This labeling strategy enables labels to be assigned automatically to a large number of ELF files. However, symbol-based labels do not directly prove that QV APIs are actually invoked along executable paths. Therefore, this study explicitly defines the labeling criteria and applies a cleaning procedure to prevent overlap between the training and evaluation data. During cleaning, files with the same binary identity or the same SHA-256 hash were removed to ensure that they did not appear in both the training and evaluation sets. In addition, data were split by group so that binaries from the same package or the same binary family did not appear in multiple splits. After this cleaning process, the final clean training set contained 1,315 samples, consisting of 656 candidate samples and 659 non-candidate samples.

4.3 Model Training and Selection

This study compares four machine learning models: Logistic Regression, SVM, Random Forest, and XGBoost. The purpose of this comparison is to determine which type of model is more suitable for screening QV candidates from the extracted features. Because symbol names cannot be used directly as numerical inputs, TF-IDF is applied to convert them into numerical features. The TF-IDF symbol features are then combined with symbol counts, library dependencies, and binary properties, and the same combined feature representation is used as input to all models.

Table 4 summarizes the compared models and the purpose of each comparison. These four models were selected because the extracted features have different characteristics. Linear models such as Logistic Regression and SVM may be effective for high-dimensional symbol-name features,

whereas tree-based models may better capture combinations among symbol counts, library dependencies, and binary properties.

Table 4. Compared machine learning models and their purposes

Model	Purpose of Comparison
Logistic Regression	To examine whether a simple classification criterion is effective for symbol-based features
SVM	To examine whether a clear decision boundary exists between candidates and non-candidates
Random Forest	To examine whether combinations of multiple features improve candidate classification
XGBoost	To examine whether more sophisticated feature interactions can be captured by a tree-based model

4.4 Inference Pipeline

After training, ML-QED-Lite stores the trained model and the configuration required for analysis in a single file. During actual analysis, this file is loaded, and ML-QED-Lite examines ELF executables in the input directory one by one. For each file, features are extracted and provided to the model, which outputs a score indicating the likelihood that the file is a PQC migration candidate. If the score is greater than or equal to a predefined threshold, the file is classified as a PQC migration candidate; otherwise, it is classified as a non-candidate. After analysis, ML-QED-Lite stores the candidate list, the score of each file, the threshold used for classification, and required library information in a `json` file.
Algorithm 1 summarizes the screening procedure.

4.5 Screening Report

After the inference pipeline in Section 4.4 determines the score and candidate decision for each file, ML-QED-Lite assigns a result tier to help analysts decide which files should be inspected first. The tier is not a separate classification result. Instead, it is determined by the distance between the score s of each file and the threshold τ. For candidate files, a higher score indicates a stronger candidate. If a file is classified as a candidate despite having no QV-related symbols, ML-QED-Lite reports the possibility of an FP. Among non-candidate files, those located immediately below the threshold are separately marked for review because they may correspond to missed candidates. Files sufficiently below the threshold are regarded as safe. These tiers do not change TP/FP decisions; they are provided only to make the output easier to interpret. The tier criteria are summarized in Table 5.

Algorithm 1 ELF inference procedure of ML-QED-Lite

Require: scan directory P, feature extractor ϕ, classifier f, threshold τ
Ensure: result report R
$\mathcal{B} \leftarrow$ collect ELF binaries under P
$\mathcal{D} \leftarrow \emptyset$
for all ELF binary $x \in \mathcal{B}$ **do**
    $r_x \leftarrow$ extract symbols, library dependencies, and binary properties from x
    add r_x to $\mathcal{D}$
end for
$X \leftarrow \phi(\mathcal{D})$
$S \leftarrow f(X)$
$R \leftarrow \emptyset$
for all record $r_x \in \mathcal{D}$ and score $s_x \in S$ **do**
    if $s_x \geq \tau$ **then**
        mark x as a PQC migration candidate
    else
        mark x as a non-candidate
    end if
    append prediction, score, threshold, required libraries, and feature summary to R
end for
emit `result.json`
return R

Table 5. Result tier criteria of ML-QED-Lite

Tier	Condition	Terminal Output Message
high	$s \geq 0.80$	strong candidate – review first
medium	$\tau + 0.10 \leq s < 0.80$	candidate – review recommended
borderline	$\tau \leq s < \tau + 0.10$	weak candidate – low confidence, verify
FN-suspect	$\tau - 0.10 \leq s < \tau$	possibly missed – review recommended
safe	$s < \tau - 0.10$	no review needed

5 Performance Evaluation

This section presents the experimental environment and evaluation results. The compared tools are QED, the QED-Lite baseline, and ML-QED-Lite. We first measure QED and QED-Lite under the same environment to examine their differences in detection accuracy and analysis cost. We then compare QED-Lite with ML-QED-Lite to evaluate whether the machine learning-based extension can reduce FPs while preserving TPs. Detection accuracy is evaluated using TP, FP, FN, TN, TPR, TNR, and FPR. Analysis cost is measured using runtime and maximum resident set size (Max RSS) memory. Runtime is reported as the mean and standard deviation over three runs, and Max RSS is reported in MB.

5.1 Experimental Setup

The experimental environment was configured using a Docker image based on Ubuntu 20.04. To approximate the environment of the original QED study, OpenSSL 3.3.1, Mbed TLS 2.28.8, wolfSSL 5.7.2-stable, tpm2-tss 3.1.0, and tpm2-tools 4.3.2 were built inside the Docker image. The main hardware and software specifications are summarized in Table 6.

Table 6. Experimental environment

Item	Version or Setting
Hardware Environment	
CPU	13th Gen Intel(R) Core(TM) i5-1340P
Logical CPUs	16
Memory	7.6 GiB
Kernel	Linux 6.6.87.2-microsoft-standard-WSL2
Software Environment	
Base OS	Ubuntu 20.04
Python	3.8.10
GCC	9.4.0
GLIBC	2.31
System OpenSSL package	1.1.1f
angr	9.2.20
capstone	4.0.2
networkx	3.1
pyelftools	0.32
scikit-learn	1.3.2
XGBoost	2.0.3
Built Libraries and Tools	
OpenSSL	3.3.1
Mbed TLS	2.28.8
wolfSSL	5.7.2-stable
tpm2-tss	3.1.0
tpm2-tools	4.3.2

5.2 Evaluation Dataset and Ground Truth

In this study, ground truth is defined based on whether each ELF executable is likely to directly use QV APIs. In the synthetic dataset, files using RSA, ECDSA, or DH are defined as QV, whereas files using only

AES or SHA512 are defined as non-QV. The coreutils dataset consists of basic Linux command-line utilities, and unixbench is a system benchmark suite. Since these programs are not intended to use public-key cryptography, they are used as non-QV evaluation data.
In the network dataset, `ssh`, `sshd`, `nmap`, `wget`, `dig`, `nslookup`, and `curl` are defined as QV. The `scp` and `sftp` executables are linked against `libcrypto`, and therefore may be detected as candidates by dependency-based tools. However, because no QV API imports such as RSA, DSA, DH, EC, or EVP_PKEY were observed, they are treated as non-QV in the ground truth.

5.3 Baseline Reproduction

Table 7 shows the results on the synthetic dataset. QED achieved TP 24, FP 0, FN 0, and TN 16, resulting in an accuracy of 100.00%. QED-Lite achieved TP 6, FP 4, FN 18, and TN 12, with an accuracy of 45.00%, a TPR of 25.00%, and an FPR of 25.00%. Thus, the accuracy of QED-Lite was 0.45 times that of QED, and its TPR was 0.25 times that of QED. Its TNR was 0.75 times that of QED.

Table 7. Accuracy of QED and QED-Lite on the synthetic dataset

Tool	TP	FP	FN	TN	Acc.	TPR	TNR	FPR	FNR
QED	24	0	0	16	100.00	100.00	100.00	0.00	0.00
QED-Lite	6	4	18	12	45.00	25.00	75.00	25.00	75.00

Table 8 shows the results on the real-world dataset, excluding tpm2-tools. QED achieved TP 4, FP 0, FN 3, and TN 133, with an accuracy of 97.86% and a TPR of 57.14%. QED-Lite achieved TP 5, FP 2, FN 2, and TN 131, with an accuracy of 97.14%, a TPR of 71.43%, and an FPR of 1.50%.
The accuracy of QED-Lite was approximately 0.99 times that of QED. Its TPR was approximately 1.25 times higher than that of QED, and its FNR decreased to approximately 0.67 times that of QED. However, QED-Lite newly introduced an FPR of 1.50%. These results indicate that QED-Lite detected more ground-truth QV files in the real-world dataset, but also incorrectly classified some non-QV files as QV candidates.

Table 8. Accuracy of QED and QED-Lite on the real-world dataset excluding tpm2-tools

Tool	TP	FP	FN	TN	Acc.	TPR	TNR	FPR	FNR
QED	4	0	3	133	97.86	57.14	100.00	0.00	42.86
QED-Lite	5	2	2	131	97.14	71.43	98.50	1.50	28.57

5.4 End-to-End Evaluation of ML-QED-Lite

Table 9 compares the four machine learning models. Logistic Regression achieved the highest AUC of 88.33% and was therefore selected as the final classifier. Although SVM achieved a similar accuracy and a slightly higher balanced accuracy, Logistic Regression was selected because AUC reflects threshold-independent separability, which is important when the threshold is later adjusted for FP control.

Table 9. Comparison of four machine learning models

Model	AUC	Th.	Acc.	TPR	TNR	FPR	FNR	Prec.	B.Acc.
Logistic Regression	88.33	0.507	81.37	66.92	95.49	4.51	33.08	93.55	81.21
SVM	86.93	0.342	81.37	70.00	92.48	7.52	30.00	90.10	81.24
Random Forest	76.69	0.725	74.14	68.46	79.70	20.30	31.54	76.72	74.08
XGBoost	84.87	0.950	77.57	56.15	98.50	1.50	43.85	97.33	77.33

All values except the threshold are reported as percentages.

Description of Evaluation Metrics

AUC	Ability to distinguish candidates from non-candidates without fixing a single threshold
Th.	Candidate decision threshold selected on the validation set
Acc.	Proportion of all samples that were correctly classified
Prec.	Proportion of actual candidates among samples predicted as candidates
B.Acc.	Average of TPR and TNR

Based on these results, the final version of ML-QED-Lite was directly compared with QED-Lite. Table 10 compares the detection accuracy of QED-Lite and ML-QED-Lite on the network and synthetic datasets.

Table 10. End-to-end accuracy comparison between QED-Lite and ML-QED-Lite

Tool	Dataset	TP	FP	FN	TN	Acc.	TPR	TNR	FPR	FNR
QED-Lite	network	5	2	2	4	69.23	71.43	66.67	33.33	28.57
***ML-QED-Lite**	network	5	0	2	6	84.62	71.43	100.00	0.00	28.57
QED-Lite	synthetic	6	4	18	12	45.00	25.00	75.00	25.00	75.00
***ML-QED-Lite**	synthetic	6	0	18	16	55.00	25.00	100.00	0.00	75.00

On the network dataset, ML-QED-Lite preserved the same five TPs as QED-Lite while reducing FPs from two to zero. On the synthetic dataset, ML-QED-Lite also preserved the same six TPs as QED-Lite while reducing FPs from four to zero. These results show that ML-QED-Lite did not detect additional QV files compared with QED-Lite, but it reduced unnecessary candidates while preserving the same TP count.

5.5 Runtime and Memory

Table 11 compares the runtime and memory usage of QED, QED-Lite, and ML-QED-Lite. On the network dataset, QED required an average runtime of 237.167 s and 3070.40 MB of Max RSS, whereas ML-QED-Lite required an average runtime of 1.113 s and 104.32 MB of Max RSS. On the synthetic dataset, QED required an average runtime of 12.920 s and 309.82 MB of Max RSS, whereas ML-QED-Lite required an average runtime of 1.917 s and 103.55 MB of Max RSS.
ML-QED-Lite incurs additional cost compared with QED-Lite because it performs feature construction and model inference. However, compared with the original QED, both its runtime and memory usage are substantially smaller. Therefore, ML-QED-Lite should be interpreted not as being as lightweight as QED-Lite, but as a screening tool that remains much lighter than QED while reducing FPs compared with QED-Lite.

Table 11. Runtime and memory of QED, QED-Lite, and ML-QED-Lite

Tool	**Dataset**	**Runs**	**Runtime** (s)	**Max RSS** (MB)
QED	coreutils	3	3.010 ± 0.070	129.85
QED	network	3	237.167 ± 0.626	3070.40
QED	synthetic	3	12.920 ± 0.123	309.82
QED	tpm2	3	254.153 ± 1.472	3424.15
QED	unixbench	3	1.607 ± 0.064	129.92
QED-Lite	coreutils	3	0.607 ± 0.021	16.45
QED-Lite	network	3	0.537 ± 0.012	18.67
QED-Lite	synthetic	3	0.397 ± 0.006	17.38
QED-Lite	tpm2	3	1.350 ± 0.053	18.15
QED-Lite	unixbench	3	0.160 ± 0.000	16.19
***ML-QED-Lite**	coreutils	3	5.187 ± 0.040	103.87
***ML-QED-Lite**	network	3	1.113 ± 0.040	104.32
***ML-QED-Lite**	synthetic	3	1.917 ± 0.070	103.55
***ML-QED-Lite**	tpm2	3	3.253 ± 0.075	103.69
***ML-QED-Lite**	unixbench	3	1.190 ± 0.010	103.68

5.6 Limitations

The main limitation of this study is the small size of the evaluation datasets. In particular, the network dataset contains only seven files defined as QV. Therefore, a single changed prediction can substantially affect the TPR. For example, changing the result of one file among seven changes the TPR by 14.29 percentage points. Thus, the network dataset results should be interpreted as preliminary evidence of the potential of

ML-QED-Lite, rather than as statistically stable generalization performance. Future work should construct larger and more diverse evaluation sets.

Second, the training labels are weak labels derived from symbols. In this study, a candidate label was assigned when QV-related APIs were identified among imported symbols. However, the presence of a symbol does not prove that the corresponding API is actually invoked along an executable path. Therefore, the labels used in this study should be interpreted as evidence of possible QV cryptographic usage, not as ground truth proving actual runtime invocation. Future work should construct more reliable labels by combining manual review, dynamic tracing, and QED-style API and static trace evidence.

6 Conclusion

This study investigated whether PQC migration candidates can be rapidly identified from ELF executables without source code while reducing FPs introduced during lightweight screening. To this end, we constructed a common experimental environment and re-evaluated QED and QED-Lite under the same conditions. The results show that QED provides the highest precision but incurs substantial analysis cost, whereas QED-Lite is significantly faster but less precise than QED and incorrectly classifies some non-QV files as candidates.

To address this limitation, we proposed ML-QED-Lite, a tool that directly classifies all ELF executables in a target directory. ML-QED-Lite extracts function symbols, library dependencies, and binary-level properties from each file and uses these features for classification. We compared four machine learning models, namely Logistic Regression, SVM, Random Forest, and XGBoost. Logistic Regression achieved the highest AUC and was selected as the final classifier.

The evaluation results show that ML-QED-Lite reduces FPs while preserving the same number of TPs as QED-Lite. On the network dataset, ML-QED-Lite detected the same five TPs as QED-Lite and reduced FPs from two to zero. On the synthetic dataset, it detected the same six TPs as QED-Lite and reduced FPs from four to zero. Although ML-QED-Lite incurs higher runtime and memory overhead than QED-Lite, it remains substantially lighter than the original QED. These results indicate that ML-QED-Lite improves the practicality of lightweight screening for PQC migration by reducing unnecessary manual inspection while preserving the candidate-detection capability of QED-Lite.

This study has several limitations. The evaluation is an initial validation based on small evaluation datasets and automatically assigned weak labels. Future work should construct larger and more independent evaluation datasets and examine the robustness of ML-QED-Lite under stripped binaries, different compiler optimizations, and diverse versions of cryptographic libraries. In addition, more reliable labels should be constructed by combining manual review, dynamic tracing, and static trace evidence. Such improvements would make ML-QED-Lite a more robust screening tool for practical PQC migration workflows.

References

1. Arakelyan, S., Arasteh, S., Hauser, C., Kline, E., Galstyan, A.: Bin2vec: Learning representations of binary executable programs for security tasks. Cybersecurity **4**(26) (2021). https://doi.org/10.1186/s42400-021-00088-4
2. Fawcett, T.: An introduction to roc analysis. Pattern Recognition Letters **27**(8), 861–874 (2006). https://doi.org/10.1016/j.patrec.2005.10.010
3. Gröbert, F., Willems, C., Holz, T.: Automated identification of cryptographic primitives in binary programs. In: Recent Advances in Intrusion Detection. Lecture Notes in Computer Science, vol. 6961, pp. 41–60. Springer (2011). https://doi.org/10.1007/978-3-642-23644-0_3
4. Hasan, K.F., Simpson, L., Baee, M.A.R., Islam, C., Rahman, Z., Armstrong, W., Gauravaram, P., McKague, M.: A framework for migrating to post-quantum cryptography: Security dependency analysis and case studies. IEEE Access **12**, 23427–23450 (2024). https://doi.org/10.1109/ACCESS.2024.3360412
5. Joseph, D., Misoczki, R., Manzano, M., Tricot, J., Pinuaga, F.D., Lacombe, O., Leichenauer, S., Hidary, J., Venables, P., Hansen, R.: Transitioning organizations to post-quantum cryptography. Nature **605**(7909), 237–243 (2022). https://doi.org/10.1038/s41586-022-04623-2
6. Kaufman, S., Rosset, S., Perlich, C.: Leakage in data mining: Formulation, detection, and avoidance. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 556–563. ACM (2011). https://doi.org/10.1145/2020408.2020496
7. Kim, H.G., Lee, S.W., Bang, J.W., Kim, U.J., Kang, H.J., Kim, M.S., Seo, H.J.: QED-lite: Lightweight detection of quantum-vulnerable ELF binaries via cryptographic library version fingerprinting. Cryptology ePrint Archive, Paper 2026/660 (2026), https://eprint.iacr.org/2026/660
8. LaMacchia, B.: The long road ahead to transition to post-quantum cryptography. Communications of the ACM **65**(1), 28–30 (2022), https://cacm.acm.org/opinion/the-long-road-ahead-to-transition-to-post-quantum-cryptography/
9. 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. pp. 203–214 (2015)
10. Näther, C., Herzinger, D., Gazdag, S.L., Steghöfer, J.P., Daum, S., Loebenberger, D.: Migrating software systems toward post-quantum cryptography: A systematic literature review. IEEE Access **12**, 132107–132126 (2024). https://doi.org/10.1109/ACCESS.2024.3450306
11. National Institute of Standards and Technology: NIST releases first 3 finalized post-quantum encryption standards (2024), https://ww

w.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards, accessed: 2026-06-05
12. Pendlebury, F., Pierazzi, F., Jordaney, R., Kinder, J., Cavallaro, L.: {TESSERACT}: Eliminating experimental bias in malware classification across space and time. In: 28th USENIX security symposium (USENIX Security 19). pp. 729–746 (2019)
13. 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
14. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science. pp. 124–134 (1994). https://doi.org/10.1109/SFCS.1994.365700
15. Xu, D., Ming, J., Wu, D.: Cryptographic function detection in obfuscated binaries via bit-precise symbolic loop mapping. In: 2017 IEEE Symposium on Security and Privacy. pp. 921–937. IEEE (2017). https://doi.org/10.1109/SP.2017.56