QORONEX LTD
Oxford, UK · 2026

PUBLIC — MIT OPEN SOURCE TOOL

CryptoBOM: Automated Cryptographic Bill of Materials for Automotive ECU Firmware

Technical Report · Version 1.1 · March 2026

CryptoBOM is an open-source command-line tool that statically analyses automotive ECU firmware binaries to produce a structured Cryptographic Bill of Materials, supporting UNECE R155 compliance and post-quantum migration planning.

REPORT METADATA

Authors: QORONEX LTD Research Team
Affiliation: QORONEX LTD, Oxford, UK
Licence: MIT Open Source Licence
GitHub: github.com/qoronex/cryptobom
Contact: info@qoronex.co.uk

Contents

Section 1 · Abstract

CryptoBOM is an open-source command-line tool that scans automotive ECU firmware binaries in ELF, Intel HEX, Motorola S-record, and raw binary formats to produce a structured Cryptographic Bill of Materials. It identifies cryptographic algorithms and library references present in a firmware image, classifies each finding as **CRITICAL**, **REVIEW**, or **SAFE** for post-quantum readiness, and writes a machine-readable Excel report. CryptoBOM is built to help automotive OEMs and Tier-1 suppliers meet UNECE R155 requirements and plan migration to NIST post-quantum algorithms. It is released under the MIT Licence and available on GitHub.

Section 2 · Motivation

2.1 Regulatory Context

UNECE Regulation No. 155 (R155) became mandatory for new vehicle type approvals in July 2022 and covers all new vehicles sold from July 2024. It requires OEMs to maintain a current inventory of cybersecurity assets across the vehicle lifecycle, with cryptographic algorithms explicitly named as assets subject to annual review. The regulation does not yet mandate post-quantum readiness, but ENISA expects PQC migration timelines to feature in future revisions. Several major OEMs have already started internal PQC programmes.

2.2 The Scalability Problem

A modern passenger vehicle contains 50 to 150 distinct ECU types, each running its own firmware stack. Manual review of cryptographic usage across a fleet of this size is neither scalable nor repeatable. For Tier-2 and Tier-3 supplied components, source code is frequently unavailable altogether, making static analysis of compiled binaries the only practical path to a complete vehicle-level crypto inventory. No publicly available tool combines automotive ECU binary format support with structured, machine-readable PQC risk classification aligned to current NIST standards.

2.3 Harvest Now, Decrypt Later

The *Harvest Now, Decrypt Later* (HNDL) threat model describes adversaries capturing and storing encrypted communications or firmware update packages today, to decrypt them once a cryptographically relevant quantum computer (CRQC) is available. Most estimates place the CRQC horizon at 10 to 15 years. HNDL is already an operational threat for data with long confidentiality requirements, and automotive firmware and OTA update keys fit that category. Finding RSA and elliptic curve Diffie-Hellman / ECDSA usage in deployed firmware is now a practical priority.

2.4 Gap in Available Tooling

- General-purpose binary analysis and reverse engineering frameworks require significant manual configuration and expert knowledge to extract crypto-specific findings at scale.
- Existing software composition analysis (SCA) tools operate on source code or package manifests, not compiled binaries.
- No open-source tool produces a machine-readable CryptoBOM aligned with automotive binary formats and NIST PQC migration guidance simultaneously.

Section 3 · Tool Architecture

CryptoBOM is written in Python 3.10+ and built as four independent layers. It runs on any platform where Python is available and can be dropped into an existing CI/CD pipeline with a single command.

INPUT	ELF · Intel HEX · S-record · Raw Binary
PARSER	Binary format detection & normalisation
ANALYSER	Signature matching + symbol resolution
REPORTER	Finding classification & risk scoring
OUTPUT	Excel report · JSON · Console terminal

Figure 1 · CryptoBOM high-level pipeline

3.1 Binary Format Detection

The input layer detects the binary format by inspecting file magic bytes and structural markers. Supported formats are ELF (32-bit and 64-bit, little- and big-endian), Intel HEX (:00000001FF terminator), Motorola S-record (S0-S9 records), and raw binary. The detected format is recorded in the output report.

3.2 Static Analysis Engine

The analyser uses two techniques. First, a signature engine scans the raw byte stream produced after stripping ELF headers or HEX record framing, looking for algorithm-specific constants such as known public-key parameters and hash initialisation vectors. Second, a symbol pass extracts library and function references and matches them against a fingerprint database covering mbedTLS, wolfSSL, OpenSSL, the Infineon TrustCore SDK, and the NXP HSM API. Results from both passes are merged and de-duplicated.

3.3 Finding Classification

Each detected algorithm receives one of three risk labels based on current NIST guidance:

[CRITICAL] — Algorithm is quantum-vulnerable; replace before CRQC horizon

[REVIEW] — Algorithm is classically weak or deprecated; evaluate urgency

[SAFE] — Algorithm has no known quantum vulnerability at current key sizes

3.4 Report Generation

The reporter writes findings to an Excel workbook with separate sheets for the scan summary, per-algorithm detail, and migration actions. A JSON file is also written for use in scripts and pipelines. Every report includes the tool version, scan timestamp, and a SHA-256 hash of the input file.

Section 4 · Detection Capabilities

4.1 Algorithm Detection Matrix

Table 1 lists each algorithm detected by CryptoBOM v1.0, its risk level, how it is detected, and the NIST-recommended post-quantum replacement.

Algorithm	Risk Level	Detection Method	NIST Replacement
ECC P-256	CRITICAL	Pattern + symbol	ML-DSA-65 (FIPS 204)
ECC P-384	CRITICAL	Pattern + symbol	ML-DSA-87 (FIPS 204)
RSA-2048	CRITICAL	Pattern + symbol	ML-KEM-768 (FIPS 203)
RSA-4096	CRITICAL	Pattern + symbol	ML-KEM-1024 (FIPS 203)
ECDH / ECDSA	CRITICAL	Symbol	ML-KEM / ML-DSA
SHA-1	REVIEW	Pattern	SHA-3 or SHA-256
MD5	REVIEW	Pattern	SHA-256
3DES	REVIEW	Pattern	AES-256
SHA-256	SAFE	Pattern	No change required
SHA-512	SAFE	Pattern	No change required
AES-128	SAFE	Pattern	No change required
AES-256	SAFE	Pattern	No change required
ChaCha20	SAFE	Pattern	No change required

Table 1 · CryptoBOM algorithm detection matrix (v1.0)

4.2 Supported Cryptographic Libraries

Symbol-level detection covers the libraries below. Together they account for most cryptographic usage seen in automotive ECU firmware:

Library	Version Range Tested	Detection Coverage
mbedTLS	2.x – 3.x	Full (pattern + symbol)
wolfSSL	4.x – 5.x	Full (pattern + symbol)
OpenSSL	1.1.x – 3.x	Full (pattern + symbol)
Infineon TrustCore SDK	2.x	Symbol-level
NXP HSM API	1.x	Symbol-level

Table 2 · Supported cryptographic library fingerprints

Section 5 · Example Output

5.1 Terminal Output

The output below is from a scan of a representative gateway ECU firmware image. Each finding shows the address, library, and risk label, with the NIST-recommended replacement and its FIPS reference.

```
$ cryptobom scan gateway_ecu_v2.3.elf

CryptoBOM v1.0  |  QORONEX LTD  |  MIT Licence
Input  : gateway_ecu_v2.3.elf  (ELF32 LE, 1.4 MB)
Scan started : 2026-03-24T09:15:32Z

Scanning gateway_ecu_v2.3.elf...

[CRITICAL] 0x4A20  ECDSA P-256   lib: mbedTLS
           -> Replace: ML-DSA-65 (FIPS 204)
[CRITICAL] 0x8B40  RSA-2048      lib: mbedTLS
           -> Replace: ML-KEM-768 (FIPS 203)
[CRITICAL] 0xC210  ECDH P-256    lib: mbedTLS
           -> Replace: ML-KEM-768 (FIPS 203)
[REVIEW]   0x1F80  SHA-1         lib: mbedTLS
           -> Replace: SHA-256 or SHA-3
[SAFE]     0x3320  AES-128       lib: mbedTLS
[SAFE]     0x3A10  SHA-256       lib: mbedTLS

████████████████████████████████████████████████████████████
Summary   : 6 findings  (3 CRITICAL  |  1 REVIEW  |  2 SAFE)
PQC-Ready : NO
Report    : gateway_ecu_v2.3_cryptobom_20260324.xlsx
Elapsed   : 2.3 s
```

Figure 2 · CryptoBOM terminal output — gateway ECU example

5.2 Excel Report Structure

The Excel workbook has three sheets. Headers for each are shown below.

Sheet: Summary

File	Scan Date	Tool Version	SHA-256	CRITICAL	REVIEW	SAFE	PQC-Ready

Sheet: Findings

Address	Algorithm	Risk Level	Library	Symbol	NIST Replacement	FIPS Ref

Sheet: Migration

Algorithm	Risk Level	Instances	NIST Replacement	FIPS Ref	Priority	Notes

Section 6 · Validation

6.1 Test Corpus

Six synthetic test binaries were used to validate CryptoBOM v1.0. Each binary contains cryptographic constants taken from published NIST FIPS standards and checked against the wolfSSL v5.7.0 source tree. None of the binaries contains proprietary vendor code; the platform names in the filenames describe the test scenario, not the source of the binary. All four supported input formats are covered: raw binary, ELF64, Intel HEX, and Motorola S-record.

Constants were taken from NIST FIPS 186-4 (P-256 parameters), NIST FIPS 180-4 (SHA-256 IV and round constants), RFC 8017 (PKCS#1 v1.5 DigestInfo), and ITU-T X.690 (DER encoding of RSA public exponent e = 65537).

6.2 Summary Results

Binary	Format	Library Simulated	CRITICAL	SAFE	Total	PQC-Ready
wolfssl_ecc_tls_cortexm.bin	Raw .bin	wolfSSL (ARM Cortex-M)	6	3	9	NO
mbedtls_rsa2048_secureboot_ppc.bin	Raw .bin	mbedTLS (PowerPC BE)	5	2	7	NO
openssl_tls13_gateway_x86.bin	ELF64	OpenSSL (x86-64 Linux)	6	1	7	NO
mbedtls_aes_only_secoc.bin	Raw .bin	mbedTLS - AES-only (SecOC)	0	1	1	YES
stm32_tfm_ecc_secureboot.hex	Intel HEX	mbedTLS (STM32L5 TFM)	5	2	7	NO
nxp_mpc5748g_autosar_crypto.srec	S-record	mbedTLS + wolfSSL (MPC5748G)	9	2	11	NO

Table 3 · CryptoBOM v1.0 verified scan results -- six synthetic automotive firmware fixtures

6.3 Per-Binary Detail

Each table below lists every finding for one test binary: address, matched rule, and NIST post-quantum replacement. All results were checked against the relevant NIST FIPS publications.

Binary 1 -- wolfssl_ecc_tls_cortexm.bin

Scenario: Zephyr HTTPS client compiled with wolfSSL (ARM Cortex-M LE) | **Format:** Raw binary | **Size:** 1,157 bytes | **PQC-Ready:** NO

CRITICAL findings (6)

#	Algorithm	Address	Matched Rule	NIST Replacement
1	ECC_P256	0x00000200	ECC_P256_Prime_P	ML-DSA-65 (FIPS 204) / ML-KEM-768 (FIPS 203)
2	RSA	0x00000224	RSA_Montgomery_Barrett_Sentinels	ML-KEM-768 (FIPS 203) / ML-DSA-65 (FIPS 204)
3	ECC_P256	0x00000240	ECC_P256_Coefficient_B	ML-DSA-65 (FIPS 204) / ML-KEM-768 (FIPS 203)
4	ECC_P256	0x00000260	ECC_P256_BasePoint_Gx	ML-DSA-65 (FIPS 204) / ML-KEM-768 (FIPS 203)
5	ECC_P256	0x00000280	ECC_P256_BasePoint_Gy	ML-DSA-65 (FIPS 204) / ML-KEM-768 (FIPS 203)
6	ECC_P256	0x00000384	ECC_P256_Library_Symbols	ML-DSA-65 (FIPS 204) / ML-KEM-768 (FIPS 203)

SAFE findings (3)

Algorithm	Address	Matched Rule

SHA256	0x000002c0	SHA256_IV_Sequence
SHA256	0x000002e0	SHA256_RoundConstants_K0_K15
SHA256	0x000003c7	SHA256_Library_Symbols

Binary 2 -- mbedtls_rsa2048_secureboot_ppc.bin

Scenario: AUTOSAR BSW RSA-2048 secure-boot image (PowerPC big-endian) | **Format:** Raw binary | **Size:** 942 bytes | **PQC-Ready:** NO

CRITICAL findings (5)

#	Algorithm	Address	Matched Rule	NIST Replacement
1	RSA	0x00000100	RSA_PKCS1_DigestInfo_SHA256	ML-KEM-768 (FIPS 203) / ML-DSA-65 (FIPS 204)
2	RSA	0x00000122	RSA_PKCS1_DigestInfo_SHA384	ML-KEM-768 (FIPS 203) / ML-DSA-65 (FIPS 204)
3	RSA	0x00000164	RSA_PublicExponent_65537	ML-KEM-768 (FIPS 203) / ML-DSA-65 (FIPS 204)
4	RSA	0x0000018c	RSA_Montgomery_Barrett_Sentinels	ML-KEM-768 (FIPS 203) / ML-DSA-65 (FIPS 204)
5	RSA	0x0000025c	RSA_Library_Symbols	ML-KEM-768 (FIPS 203) / ML-DSA-65 (FIPS 204)

SAFE findings (2)

Algorithm	Address	Matched Rule
SHA256	0x000001fc	SHA256_IV_Sequence
SHA256	0x000002df	SHA256_Library_Symbols

Binary 3 -- openssl_tls13_gateway_x86.bin

Scenario: Automotive Linux gateway using OpenSSL for TLS 1.3 (x86-64 LE) | **Format:** ELF64 | **Size:** 2,168 bytes | **PQC-Ready:** NO

CRITICAL findings (6)

#	Algorithm	Address	Matched Rule	NIST Replacement
1	ECC_P256	0x00000080	ECC_P256_Prime_P	ML-DSA-65 (FIPS 204) / ML-KEM-768 (FIPS 203)
2	RSA	0x00000084	RSA_Montgomery_Barrett_Sentinels	ML-KEM-768 (FIPS 203) / ML-DSA-65 (FIPS 204)
3	ECC_P256	0x000000a0	ECC_P256_BasePoint_Gx	ML-DSA-65 (FIPS 204) / ML-KEM-768 (FIPS 203)
4	ECC_P256	0x000000c0	ECC_P256_BasePoint_Gy	ML-DSA-65 (FIPS 204) / ML-KEM-768 (FIPS 203)
5	ECC_P256	0x00000140	ECC_P256_Library_Symbols	ML-DSA-65 (FIPS 204) / ML-KEM-768 (FIPS 203)
6	RSA	0x000001a3	RSA_Library_Symbols	ML-KEM-768 (FIPS 203) / ML-DSA-65 (FIPS 204)

SAFE findings (1)

Algorithm	Address	Matched Rule
SHA256	0x000000e0	SHA256_IV_Sequence

Binary 4 -- mbedtls_aes_only_secoc.bin [PQC-READY]

Scenario: AUTOSAR SecOC ECU -- AES-GCM and SHA-256 only, no asymmetric crypto | **Format:** Raw binary | **Size:** 752 bytes | **PQC-Ready:** YES

CRITICAL findings: NONE

SAFE findings (1)

Algorithm	Address	Matched Rule
SHA256	0x000001a1	SHA256_Library_Symbols

Binary 5 -- stm32_tfm_ecc_secureboot.hex

Scenario: STM32L5 Trusted Firmware-M secure-boot using ECC P-256 | **Format:** Intel HEX | **Size:** 2,900 bytes encoded / 1,019 bytes decoded | **PQC-Ready:** NO

CRITICAL findings (5)

#	Algorithm	Address	Matched Rule	NIST Replacement
1	ECC_P256	0x00000200	ECC_P256_Prime_P	ML-DSA-65 (FIPS 204) / ML-KEM-768 (FIPS 203)
2	ECC_P256	0x00000220	ECC_P256_Coefficient_B	ML-DSA-65 (FIPS 204) / ML-KEM-768 (FIPS 203)
3	ECC_P256	0x00000240	ECC_P256_BasePoint_Gx	ML-DSA-65 (FIPS 204) / ML-KEM-768 (FIPS 203)
4	ECC_P256	0x00000260	ECC_P256_BasePoint_Gy	ML-DSA-65 (FIPS 204) / ML-KEM-768 (FIPS 203)
5	ECC_P256	0x000002a0	ECC_P256_Library_Symbols	ML-DSA-65 (FIPS 204) / ML-KEM-768 (FIPS 203)

SAFE findings (2)

Algorithm	Address	Matched Rule
SHA256	0x00000280	SHA256_IV_Sequence
SHA256	0x0000032c	SHA256_Library_Symbols

Binary 6 -- nxp_mpc5748g_autosar_crypto.srec

Scenario: NXP MPC5748G AUTOSAR BSW -- combined ECC P-256 and RSA-2048 scenario | **Format:** Motorola S-record | **Size:** 2,336 bytes encoded / 922 bytes decoded | **PQC-Ready:** NO

CRITICAL findings (9)

#	Algorithm	Address	Matched Rule	NIST Replacement
1	ECC_P256	0x00000100	ECC_P256_Prime_P	ML-DSA-65 (FIPS 204) / ML-KEM-768 (FIPS 203)
2	RSA	0x00000104	RSA_Montgomery_Barrett_Sentinels	ML-KEM-768 (FIPS 203) / ML-DSA-65 (FIPS 204)
3	ECC_P256	0x00000120	ECC_P256_Coefficient_B	ML-DSA-65 (FIPS 204) / ML-KEM-768 (FIPS 203)
4	ECC_P256	0x00000140	ECC_P256_BasePoint_Gx	ML-DSA-65 (FIPS 204) / ML-KEM-768 (FIPS 203)
5	ECC_P256	0x00000160	ECC_P256_BasePoint_Gy	ML-DSA-65 (FIPS 204) / ML-KEM-768 (FIPS 203)
6	RSA	0x000001e0	RSA_PKCS1_DigestInfo_SHA256	ML-KEM-768 (FIPS 203) / ML-DSA-65 (FIPS 204)
7	RSA	0x000001f3	RSA_PublicExponent_65537	ML-KEM-768 (FIPS 203) / ML-DSA-65 (FIPS 204)
8	RSA	0x0000020b	RSA_Library_Symbols	ML-KEM-768 (FIPS 203) / ML-DSA-65 (FIPS 204)
9	ECC_P256	0x0000028e	ECC_P256_Library_Symbols	ML-DSA-65 (FIPS 204) / ML-KEM-768 (FIPS 203)

SAFE findings (2)

Algorithm	Address	Matched Rule
SHA256	0x00000180	SHA256_IV_Sequence
SHA256	0x000001a0	SHA256_RoundConstants_K0_K15

6.4 Key Findings

• ECC P-256 was detected in 4 out of 6 test binaries. The fixtures represent scenarios spanning ARM Cortex-M, PowerPC, x86-64, and NXP MPC5748G platforms, indicating that ECC P-256 is a common cryptographic choice across diverse ECU architectures — consistent with its established prevalence in embedded TLS and secure-boot implementations.

• RSA (detected via PKCS#1 patterns, public exponent 65537, and Montgomery/Barrett constants) appeared in 3 out of 6 binaries, co-located with ECC in secure-boot and TLS scenarios.

• Binary 4, the AES-only SecOC fixture, returned zero CRITICAL findings and exit code 0. PQC-ready firmware passes through the CI/CD gate cleanly.

• All three binaries containing both ECC and RSA (Binaries 1, 3, 6) were correctly classified CRITICAL with no missed findings.

• All four input formats -- raw binary, ELF64, Intel HEX, and Motorola S-record -- were parsed and scanned without error.

• Total corpus findings: 31 CRITICAL, 11 SAFE. Zero false positives against the AES-only control binary.

6.5 Reproducibility

All six test binaries and their expected JSON reports are in the repository under **tests/test_binaries/**. To verify a result, re-scan the binary and compare the output against the reference JSON. The binary generator script is also included, so every embedded constant can be traced to its NIST or RFC source.

```
# Install CryptoBOM
pip install cryptobom

# Scan each test binary
cryptobom scan wolfssl_ecc_tls_cortexm.bin
cryptobom scan mbedtls_rsa2048_secureboot_ppc.bin
cryptobom scan openssl_tls13_gateway_x86.bin
cryptobom scan mbedtls_aes_only_secoc.bin
cryptobom scan stm32_tfm_ecc_secureboot.hex
cryptobom scan nxp_mpc5748g_autosar_crypto.srec
```

Section 7 · Availability

7.1 Open-Source Distribution

GitHub Repository	github.com/qoronex/cryptobom
Licence	MIT Open Source Licence
Language	Python 3.10+
Dependencies	yara-python, openpyxl, pyelftools, intelhex
Issue Tracker	GitHub Issues (github.com/qoronex/cryptobom/issues)
Latest Release	v1.0 — March 2026

7.1a Applicability Beyond Automotive

CryptoBOM was built for automotive ECU firmware, but the detection engine works on any binary. ICS firmware, IoT device images, and embedded Linux binaries all scan without any changes to the tool. ELF, Intel HEX, and Motorola S-record are standard formats across aerospace, defence, and medical devices, so the tool is immediately useful in those sectors as well.

7.2 Installation

```
# Install from PyPI (recommended)
$ pip install cryptobom

# Or clone from GitHub
$ git clone https://github.com/qoronex/cryptobom
$ cd cryptobom && pip install -e .

# Run a scan
$ cryptobom scan firmware.elf
```

7.3 Enterprise Edition

An enterprise edition of CryptoBOM is available for organisations with advanced compliance and fleet-scale requirements. For enquiries, please contact **info@qoronex.co.uk**.

Section 8 · Conclusion

The algorithms protecting ECU firmware, secure boot, and OTA update pipelines today will be broken by quantum computers within a foreseeable window. The problem is hard to tackle at scale: a single vehicle programme may involve hundreds of ECU variants, Tier-2 and Tier-3 components often come without source code, and no standard tooling exists to inventory cryptography at the binary level.

CryptoBOM was built by QORONEX LTD to close this gap. It scans compiled firmware directly, with no source code or build system needed, and produces a structured Cryptographic Bill of Materials with NIST-aligned PQC risk ratings. No comparable open-source tool targets automotive ECU binary formats and outputs machine-readable PQC guidance in the same step.

Section 6 shows correct results across all four binary formats and six ECU scenarios, with no false positives against the PQC-ready control binary. CryptoBOM is MIT-licensed and free to use. QORONEX LTD maintains the tool and takes contributions via GitHub. An enterprise edition is available for teams with larger compliance or fleet-scanning needs.

QORONEX LTD · Oxford, UK · info@qoronex.co.uk · github.com/qoronex/cryptobom