ÉCOLE NATIONALE SUPÉRIEURE D'ARTS ET MÉTIERS
UNIVERSITÉ HASSAN II DE CASABLANCA

Master 2 - Big Data and IoT
ENSAM Casablanca | 2025-2026

Research Progress Report

Post-Quantum Cryptography Technologies for IoT Devices

Student: Abdessamad JAOUAD
Supervisor: Prof. Ibrahim GUELZIM

December 2025

1 Research Objective

This research project aims to **survey, catalog, and analyze** the latest Post-Quantum Cryptography (PQC) technologies available for resource-constrained IoT devices. The goal is to produce a comprehensive reference document that:

- Identifies all viable PQC algorithms standardized or proposed for IoT
- Documents available libraries and tools for implementation
- Compares their characteristics, trade-offs, and suitability for small devices
- Provides a structured resource for researchers and developers entering this field

Note: This is a *survey/research* project — the deliverable is a well-structured report documenting findings, not a software implementation.

2 PQC Technologies Identified

2.1 NIST-Standardized Algorithms (August 2024)

Algorithm	Type	Standard	IoT Suitability
ML-KEM (Kyber)	Key Encapsulation	FIPS 203	High
ML-DSA (Dilithium)	Digital Signature	FIPS 204	Medium
SLH-DSA (SPHINCS+)	Digital Signature	FIPS 205	Low (large signatures)

Table 1: NIST PQC standards finalized in 2024

2.2 Additional Candidates Under Evaluation

- **Falcon** – Compact signatures, but requires floating-point (challenging for MCUs)
- **BIKE, HQC, Classic McEliece** – Code-based alternatives (Round 4 candidates)
- **FrodoKEM** – Conservative LWE-based, larger keys but simpler assumptions

3 Programming Languages Used by PQC Community

Based on analysis of major PQC projects, the cryptographic community predominantly uses:

Project	Primary Languages	Purpose
`liboqs`	C (67.9%), Assembly (30.7%)	Reference library
`PQClean`	C (51.1%), Assembly (48.3%)	Clean implementations
`pqm4`	C (64.4%), Assembly (34.1%)	ARM Cortex-M4 optimized

Table 2: Language breakdown of major PQC projects (from GitHub)

Key observation: PQC engineers chose **C and Assembly** as primary languages for performance-critical cryptographic code. Higher-level language bindings (Python, Rust, Go, Java) are provided as wrappers.

4 Reference Libraries & Implementations

4.1 Core C Libraries (Industry Standard)

Library	Description	Algorithms
`liboqs`	Open Quantum Safe C library; 117 contributors, Linux Foundation backed	ML-KEM, ML-DSA, Falcon, SPHINCS+, HQC, BIKE
`PQClean`	Clean, portable C99 implementations for integration	Kyber, Dilithium, Falcon, SPHINCS+
`pqm4`	ARM Cortex-M4 optimized (EU PQCRYPTO project)	All NIST finalists
`pq-crystals`	Official Kyber/Dilithium reference code	ML-KEM, ML-DSA

Table 3: Core C libraries used by PQC institutions

4.2 Language Bindings & Wrappers

Language	Library	Notes
Rust	`pqcrypto`, `liboqs-rust`	Auto-generated wrappers from PQClean; memory-safe
Python	`liboqs-python`	Bindings to liboqs C library
Go	`liboqs-go`	Go bindings for liboqs
Java	`liboqs-java`	JNI bindings for liboqs
C++	`liboqs-cpp`	C++ wrapper for liboqs
C#/.NET	`liboqs-dotnet`	.NET bindings
JavaScript	`node-pqclean`	Node.js/WebAssembly interface

Table 4: Language bindings provided by Open Quantum Safe project

4.3 Embedded/IoT-Specific Libraries

Library	Target Platform	Features
`pqm4`	ARM Cortex-M4 (STM32)	Assembly-optimized NTT, benchmarking framework
`wolfSSL`	Embedded/IoT	TLS 1.3 with Kyber/Dilithium support
`mbed TLS`	ARM Cortex-M	Experimental PQC branch
`liboqs + Zephyr`	Zephyr RTOS	Cross-compilation support

Table 5: Libraries targeting constrained IoT devices

5 Key Findings from Literature Review

1. **ML-KEM-512** is the most IoT-friendly: 800B public key, 768B ciphertext, fast NTT-based operations
2. **Memory constraints** are the primary barrier: Kyber requires 2KB RAM minimum; Dilithium needs 4KB
3. **Hybrid schemes** (classical + PQC) are recommended during the transition period for backward compatibility
4. **Hardware acceleration** (e.g., NTT coprocessors) significantly improves performance on Cortex-M4+
5. **Side-channel resistance** is critical: constant-time implementations are mandatory for IoT deployments

6 Resources Collected

6.1 NIST Official Documents

- FIPS 203 – ML-KEM Standard *resources/nist-fips-203-ml-kem.pdf*
- FIPS 204 – ML-DSA Standard *resources/nist-fips-204-ml-dsa.pdf*
- PQC Migration Guidelines *resources/pqc-migration-project-description-final.pdf*

6.2 Academic Surveys & Foundational Papers

- Peikert – "A Decade of Lattice Cryptography" *resources/lattice-survey.pdf*
- Regev – "The Learning with Errors Problem" *resources/lwesurvey.pdf*
- Bernstein & Lange – "Post-Quantum Cryptography" *resources/qcrypto.pdf*
- Preskill – "Quantum Computing in the NISQ Era" *resources/arxiv-1801.00862-preskill-nisq.pdf*

6.3 Online References

- NIST PQC Project: https://csrc.nist.gov/projects/post-quantum-cryptography
- Open Quantum Safe: https://openquantumsafe.org
- pqm4 Benchmarks: https://github.com/mupq/pqm4
- PQClean: https://github.com/PQClean/PQClean

7 Next Steps

1. **Expand Technology Survey**
   - Research NIST Round 4 candidates (BIKE, HQC, Classic McEliece)
   - Investigate hardware-accelerated PQC solutions (FPGA, ASIC)
   - Survey commercial IoT products with PQC support
2. **Benchmark Data Collection**
   - Gather published benchmarks from pqm4, liboqs for comparison tables
   - Document memory/speed trade-offs per algorithm per platform
3. **Report Structuring**
   - Organize findings into a comprehensive reference document
   - Create decision flowcharts for algorithm selection
   - Write practical guidelines for IoT developers
4. **Final Deliverable**
   - Complete survey report with all technologies, libraries, and recommendations
   - Include comparison tables, references, and getting-started guides

---

Repository: https://github.com/abdessamadjaouad/PQC-Algorithms