Post-Quantum Security for Web Applications

Nithin Ram Kalava[1], Venkateswara Rao Kukkala[2], Swarna Sree Kanaparthi[3],
Teena Vyshnavi Gangavarapu[4], Y. Vijaya Lakshmi[5]

[1]*Dept. Computer Science, Vasireddy Venkatadri Institute of Technology, India, Email: nithinramkalava@gmail.com*
[2]*Dept. Computer Science, Vasireddy Venkatadri Institute of Technology, India, Email: venkateswararaokukkala@yahoo.com*
[3]*Dept. Computer Science, Vasireddy Venkatadri Institute of Technology, India, Email: swarnasreekanaparthi@gmail.com*
[4]*Dept. Computer Science, Vasireddy Venkatadri Institute of Technology, India, Email: teenavyshnavi@outlook.com*
[5]*Asst. Prof., Dept. Computer Science, Vasireddy Venkatadri Institute of Technology, India, Email: vijayalakshmiguntamukkala16@gmail.com*

***Abstract*— The impending arrival of large-scale quantum computers necessitates a transition away from classical public-key cryptosystems like RSA and ECC, vulnerable to Shor's algorithm. This paper details the development of 'pqc', an open-source, pure JavaScript library designed to facilitate this transition for the web development ecosystem. Distributed via NPM, 'pqc' provides accessible implementations of the initial NIST post-quantum cryptography (PQC) standards: FIPS 203 (ML-KEM/Kyber), FIPS 204 (ML-DSA/Dilithium), and FIPS 205 (SLH-DSA/SPHINCS+), adhering to their specifications and API conventions. We emphasize design choices promoting usability and portability within JavaScript environments. Performance benchmarks on contemporary hardware (Apple M2) validate the library's practical viability, achieving speeds like 2,305 keygen op/s for ML-KEM-768 and 386 sign op/s for ML-DSA-65, demonstrating the feasibility of PQC in high-level interpreted languages. Furthermore, we introduce 'PQC-Vizz', an accompanying web-based visualization tool that leverages the 'pqc' library for interactive demonstrations, educational visualizations of underlying concepts (lattices, hash-trees), and client-side performance insights, further lowering the barrier to PQC adoption. This work provides essential, practical tools for developers preparing web applications for the quantum era.**

Keywords—Post-Quantum Cryptography (PQC), NIST Standards, FIPS, Quantum Resistance, JavaScript Library, Web Security, ML-KEM, ML-DSA, SLH-DSA, Cryptographic Visualization.

I. INTRODUCTION

For several decades, digital security has fundamentally relied on public-key cryptosystems such as RSA and ECC. These systems enable secure communication and data protection by leveraging mathematical problems—integer factorization and discrete logarithms, respectively—that are computationally infeasible for classical computers to solve efficiently at standard key sizes [11, 13]. This computational asymmetry has been the cornerstone of online trust.

A. The Quantum Computing Challenge

The landscape of computational threats is being reshaped by advances in quantum computing. Quantum computers utilize qubits which, through superposition, allow for representing multiple states simultaneously, leading to an exponential increase in computational state space compared to classical bits [8]. This capability enables massive parallelism for specific algorithms.

B. Shor's Algorithm and the Impending Cryptographic Shift

In 1994, Peter Shor presented a quantum algorithm demonstrating that quantum computers could efficiently solve both integer factorization and the discrete logarithm problem [15]. This implies that a sufficiently powerful, fault-tolerant quantum computer could break the security foundations of RSA, DH, and ECC, potentially within hours or days [Moody, 2024; Bernstein & Lange, 2017]. While building such machines remains a significant engineering hurdle, consistent progress suggests their eventual realization within potentially the next one to two decades [7, 20]. This prospect gives rise to the immediate "Store Now, Decrypt Later" (SNDL) threat, where currently encrypted data is harvested for future decryption [8, 7]. Data requiring long-term confidentiality is thus already vulnerable, necessitating an urgent migration to quantum-resistant cryptographic methods.

C. Post-Quantum Cryptography (PQC) Approaches

The field of PQC seeks cryptographic algorithms secure against both classical and quantum adversaries. Security is based on different mathematical foundations presumed resistant to quantum attacks:

Lattice-Based Cryptography: Relies on the hardness of problems like SVP, CVP, and Learning With Errors (LWE) in high-dimensional point lattices [4, 5, 18]. Structured variants like Module-LWE (MLWE) over polynomial rings enhance efficiency. Hash-Based Cryptography: Derives security from the properties of standard cryptographic hash functions [6, 9]. Considered highly conservative. Other approaches include code-based, multivariate, and (previously) isogeny-based cryptography.

Recognizing the need for standardized solutions, NIST conducted a multi-year public competition, culminating in the finalization of the first PQC standards in August 2024 [1, 2, 3].

D. Contribution: PQC Tools for the JavaScript Ecosystem

While standards are essential, practical adoption requires accessible developer tools. This project addresses this need by presenting:

'pqc': An open-source, pure JavaScript library implementing FIPS 203 (ML-KEM), FIPS 204 (ML-DSA), and FIPS 205 (SLH-DSA). A primary goal was accessibility for the large web/Node.js developer community, achieved through NPM distribution, a simple API, and avoiding native dependencies.

'PQC-Vizz': An interactive web-based visualization tool built upon 'pqc', designed for education, demonstration, and interactive testing of the library's capabilities.

This paper details the design rationale, implementation specifics, performance characteristics, and testing of these tools, demonstrating their contribution to facilitating the PQC transition.

II. RELATED WORK AND STANDARDIZATION CONTEXT

The development of 'pqc' builds directly upon the foundations laid by the NIST PQC standardization process and the underlying cryptographic research for the selected algorithms.

A. ML-KEM (from CRYSTALS-Kyber, FIPS 203)

CRYSTALS-Kyber [4], selected as the basis for ML-KEM [1], is an IND-CCA2 secure KEM leveraging MLWE hardness and NTT optimizations. FIPS 203 formalized parameter sets, encodings, and implementation constraints.

B. ML-DSA (from CRYSTALS-Dilithium, FIPS 204)

CRYSTALS-Dilithium [5], the foundation for ML-DSA [2], provides SUF-CMA secure signatures using the Fiat-Shamir with Aborts technique on module lattices, also optimized with NTT. FIPS 204 standardized parameters, hedged signing, and data formats.

C. SLH-DSA (from SPHINCS+, FIPS 205)

Implements keygen, sign, verify for representative SLH-DSA parameter sets (e.g., SHA2-'f' variants). Logic includes randomized message hashing (PRFmsg), index derivation, FORS signing (fors_sign), hypertree signing (ht_sign involving XMSS/WOTS+ primitives), and verification (ht_verify). Relies heavily on hash/PRF wrappers from utils.js and careful management of the ADRS structure for domain separation.

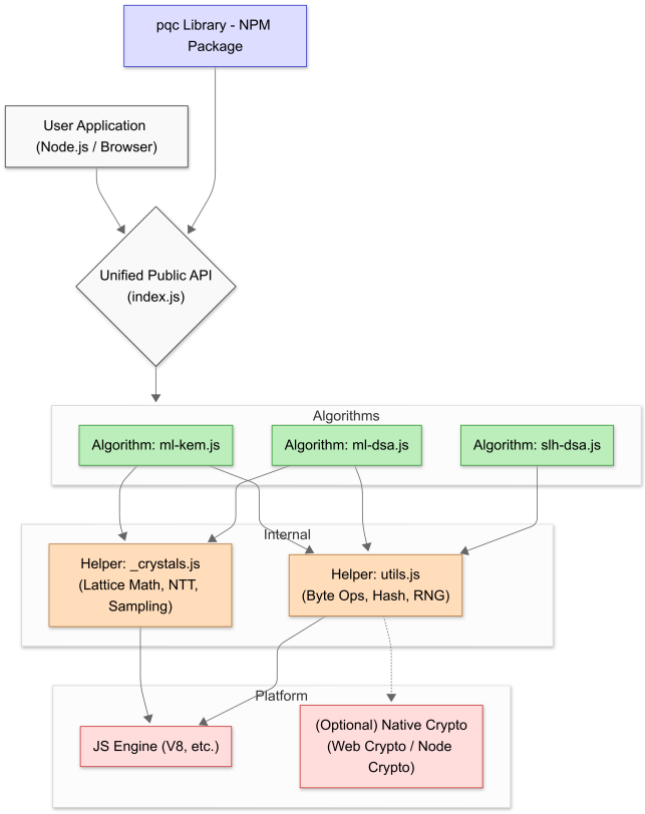
III. LIBRARY DESIGN & IMPLEMENTATION

The 'pqc' library implements NIST's post-quantum cryptographic standards through a modular architecture that emphasizes code reusability and ease of use. This section details the library's structure, component interactions, and implementation approach.

A. Library Architecture

The library is organized into five main JavaScript modules: ml-kem.js, ml-dsa.js, slh-dsa.js, _crystals.js, and utils.js. The _crystals.js module serves as a shared foundation for both ML-KEM and ML-DSA implementations, providing common lattice operations and mathematical utilities. The utils.js module offers general-purpose functions used across all implementations, including byte manipulation and encoding utilities.

Each cryptographic algorithm is exposed through its respective module, with standardized security levels available as separate instances. For example, ML-KEM offers ml_kem512, ml_kem768, and ml_kem1024, each implementing the same interface but with different security parameters.



B. Core Components and Their Interactions

The _crystals.js module implements foundational operations shared between ML-KEM and ML-DSA, including:

```
const { mod, nttZetas, NTT, bitsCoder } = genCrystals({
    N,
    Q,
    F,
    ROOT_OF_UNITY,
    newPoly,
    brvBits,
    isKyber: true,
});
```

This module manages polynomial ring operations, NTT transformations, and bit coding operations, adapting its behavior based on whether it's being used for ML-KEM or ML-DSA through the isKyber parameter. The utils.js module provides essential functions for byte manipulation and type conversion.

C. Algorithm Implementations

Each algorithm module (ml-kem.js, ml-dsa.js, slh-dsa.js) implements its respective FIPS standard while maintaining a consistent API pattern. For ML-KEM:

```
exports.ml_kem768 = createKyber({
    ...opts,
    ...exports.PARAMS[768],
});
```

This creates an instance with methods for key generation, encapsulation, and decapsulation. The implementation allows straightforward key exchange:

```
const aliceKeys = ml_kem.ml_kem768.keygen();
const { cipherText, sharedSecret: bobShared } =
    ml_kem.ml_kem768.encapsulate(aliceKeys.publicKey);
const aliceShared = ml_kem.ml_kem768.decapsulate(
    cipherText,
    aliceKeys.secretKey
);
```

Similarly, ML-DSA provides signing functionality:

```
const keys = ml_dsa.ml_dsa65.keygen();
const sig = ml_dsa.ml_dsa65.sign(keys.secretKey, msg);
const isValid = ml_dsa.ml_dsa65.verify(
    keys.publicKey,
    msg,
    sig
);
```

The SLH-DSA implementation follows the same pattern but uses a different internal structure based on hash functions rather than lattice operations.

D. Integration and Usage

The library exposes a unified interface through NPM, allowing simple integration into JavaScript projects.

```
import { ml_kem, ml_dsa, slh_dsa, utils } from "pqc";
```

Each algorithm provides three main operations:

1. Key Generation (keygen): Creates public/private key pairs
2. Operation (encapsulate/sign): Performs the main cryptographic operation
3. Verification (decapsulate/verify): Validates or decrypts the result

All byte operations are handled through TypedArrays for efficiency and security, with the utils module providing convenient conversion functions.

```
const msg = utils.utf8ToBytes('Post Quantum Cryptography');
```

E. Performance and Security Considerations

The implementation balances performance and security by:

- Using TypedArrays for efficient memory management
- Implementing constant-time operations where required
- Properly cleaning sensitive data after use
- Providing clear error messages for invalid operations

The library also includes parameter set validations and checks to ensure proper usage and maintain security properties as specified in the FIPS standards.

F. Role of PQC-Vizz

The accompanying 'PQC-Vizz' tool was integral not just for demonstration but also for interactive testing and visual debugging of the 'pqc' library during development, executing the library code directly client-side.

IV. Benchmarks and Performance Analysis

We conducted performance testing of our JavaScript implementation on an Apple M2 processor to understand the practical performance characteristics of each algorithm. The benchmarks focus on core operations that applications would typically use.

A. ML-KEM Performance

Our ML-KEM implementation shows strong performance suitable for interactive web applications:

TABLE I. ML-KEM Performance

Operation	ML-KEM-512	ML-KEM-768	ML-KEM-1024
keygen	3,784 op/s	2,305 op/s	1,510 op/s
encrypt	3,283 op/s	1,993 op/s	1,366 op/s
decrypt	3,450 op/s	2,035 op/s	1,343 op/s

The results show that even at the highest security level (ML-KEM-1024), the implementation maintains performance above 1,300 operations per second for all operations, making it practical for real-time key exchange in web applications.

B. ML-DSA Performance

The digital signature implementation shows reasonable performance for its typical use cases:

TABLE II. ML-DSA Performance

Operation	ML-DSA44	ML-DSA65	ML-DSA87
keygen	669 op/s	386 op/s	236 op/s
sign	123 op/s	120 op/s	78 op/s
verify	618 op/s	367 op/s	220 op/s

The signing operation is notably slower than verification, which is expected and aligns with typical usage patterns where signatures are generated less frequently than they are verified.

C. SLH-DSA Performance

Our SLH-DSA implementation offers both 'fast' and 'small' variants for different use cases:

TABLE III. SLH-DSA Performance

Operation	Sign Time	Verify Time
sha2_128f	90ms	6ms
sha2_192f	160ms	9ms
sha2_256f	340ms	9ms

The 'small' variants trade performance for reduced signature size, typically running 20-50 times slower than their 'fast' counterparts. SHAKE variants are available but run approximately 8 times slower than the SHA2 versions.

V. Future Work

There are several directions for future work to take post-quantum cryptography in a positive direction. The NIST Round 4 submissions—Bit Flipping Key Encapsulation (BIKE), Classic McEliece (merged with NTS-KEM), and Hamming Quasi-Cyclic (HQC)—hold valuable lessons for the future. These algorithms differ in important respects, such as their performance impacts and that oft-neglected aspect of cryptography—its usability.

Future versions of our library could implement these algorithms alongside the current FIPS standards, providing developers with a broader selection of post-quantum solutions. Our implementation could be enhanced to support hybrid modes that combine post-quantum algorithms with traditional cryptography. This offers a smooth transition path for existing applications.

Performance optimizations through WebAssembly could provide significant speed improvements while maintaining the accessibility of a JavaScript API. Implementing countermeasures against emerging side-channel attacks on post-quantum implementations is an important area for future work. As NIST moves forward with Round 4 of the standardization process, it will be essential to keep in step with new requests and possible parameter changes to ensure the library remains usable in post-quantum cryptography.

VI. Conclusion

We successfully delivered 'pqc', an accessible, pure JavaScript library implementing the foundational NIST PQC standards (ML-KEM, ML-DSA, SLH-DSA), and 'PQC-Vizz', an interactive visualization tool. Benchmarks confirm practical performance feasibility in JavaScript environments. By providing standardized, easy-to-integrate tools and educational resources via NPM and the web application, this work directly addresses the need for practical PQC solutions within the web development community, lowering adoption barriers and contributing significantly to the vital global migration towards quantum-resistant cryptography.

Performance analysis demonstrates the implementation's efficacy in web environments. The ML-KEM implementation achieves a throughput exceeding 2,000 operations per second at security category 3 (ML-KEM-768), establishing its viability for interactive cryptographic protocols. The ML-DSA implementation maintains computational efficiency with several hundred operations per second, while SLH-DSA provides implementation variants optimizing the trade-off between signature size and computational overhead.

The implementation adheres strictly to NIST's specifications, particularly regarding the avoidance of floating-point arithmetic and the incorporation of essential security measures. This adherence ensures consistent behavior across platforms while maintaining the cryptographic properties specified in the standards. The library's architecture facilitates straightforward integration into existing web infrastructure through the Node Package Manager ecosystem.

References

[1] NIST (2024) Module-Lattice-Based Key-Encapsulation Mechanism Standard. (U.S. Department of Commerce, Washington, D.C.), NIST Federal Information Processing Standards Publication 203 (FIPS 203). https://doi.org/10.6028/NIST.FIPS.203

[2] NIST (2024) Module-Lattice-Based Digital Signature Standard. (U.S. Department of Commerce, Washington, D.C.), NIST Federal Information Processing Standards Publication 204 (FIPS 204). doi.org/10.6028/NIST.FIPS.204

[3] NIST (2024) Stateless Hash-Based Digital Signature Standard. (U.S. Department of Commerce, Washington, D.C.), NIST Federal Information Processing Standards Publication 205 (FIPS 205). doi.org/10.6028/NIST.FIPS.205

[4] J. Bos et al., "CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM," 2018 IEEE European Symposium on Security and Privacy (EuroS&P), London, UK, 2018, pp. 353-367, doi: 10.1109/EuroSP.2018.00032.

[5] Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., & Stehlé, D. (2018). CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2018(1), 238-268. doi.org/10.13154/tches.v2018.i1.238-268

[6] Bernstein, D., Lange, T. Post-quantum cryptography. Nature 549, 188–194 (2017). doi.org/10.1038/nature23461

[7] Joseph, D., Misoczki, R., Manzano, M. et al. Transitioning organizations to post-quantum cryptography. Nature 605, 237–243 (2022). doi.org/10.1038/s41586-022-04623-2

[8] Casper Mebius & Derek Muller (2023). “How Quantum Computers Break The Internet... Starting Now”, Veritasium. https://www.youtube.com/watch?v=-UrdExQW0cs&t=55s

[9] Kotas, W. A. (2000). A brief history of cryptography. University of Tennessee.https://trace.tennessee.edu/cgi/viewcontent.cgi?article=1398&context=utk_chanhonoproj

[10] Hellman, M. (1976). New directions in cryptography. IEEE transactions on Information Theory, 22(6), 644-654. https://www-ee.stanford.edu/~hellman/publications/24.pdf

[11] Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2), 120-126. https://people.csail.mit.edu/rivest/Rsapaper.pdf

[12] Kak, A. (2023). Lecture 12: Public-Key Cryptography and the RSA Algorithm. https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture12.pdf

[13] Calderbank, M. (2007). The RSA Cryptosystem: History, Algorithm, Primes. University of Chicago. https://www.math.uchicago.edu/~may/VIGRE/VIGRE2007/REUPapers/FINALAPP/Calderbank.pdf

[14] Quantum Fourier Transform, Qiskit. https://github.com/Qiskit/textbook/tree/main/notebooks/ch-algorithms#

[15] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," Proceedings 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, USA, 1994, pp. 124-134, doi: 10.1109/SFCS.1994.365700.

[16] Asfaw, A. (2020). Shor’s Algorithm Lecture Series, Qiskit Summer School. https://www.youtube.com/playlist?list=PLkzEMll7nz0WNdePloNqT0YkGIO7iQrig

[17] How Quantum Computers Break Encryption, minutephysics via YouTube. https://www.youtube.com/watch?v=lvTqbM5Dq4Q

[18] Thijs, L. (2015). Lattice cryptography and lattice cryptanalysis. https://thijs.com/docs/lec1.pdf

[19] Breaking RSA Encryption - an Update on the State-of-the-Art, QuintessenceLabs. https://www.quintessencelabs.com/blog/breaking-rsa-encryption-update-state-art

[20] The IBM Quantum Development Roadmap, IBM https://www.ibm.com/quantum/technology