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 emergence of quantum computing presents an unprecedented threat to current cryptographic systems. With sufficient quantum computing capabilities, Shor's algorithm will efficiently break widely used public-key cryptography like Rivest-Shamir-Adleman (RSA) and Elliptic Curve Cryptography (ECC). To counter this threat, the National Institute of Standards and Technology (NIST) has standardized three quantum-resistant algorithms in their Federal Information Processing Standards (FIPS): Module-Lattice-based Key Encapsulation Mechanism (ML-KEM) for key encapsulation, Module-Lattice-based Digital Signature Algorithm (ML-DSA) for digital signatures, and Stateless Hash-based Digital Signature Algorithm (SLH-DSA) for hash-based signatures. This paper presents 'pqc', a JavaScript implementation of these standards available as a Node Package Manager (npm) package. Our implementation achieves practical performance on modern hardware, with ML-KEM-768 performing 2,305 operations per second for key generation and ML-DSA65 achieving 386 operations per second for signing operations. The library enables developers to integrate post-quantum cryptography into web applications, helping prepare current systems for the quantum computing era.**

Keywords—post-quantum, cryptography, NIST, PQC, FIPS

I. Introduction

The foundation of modern internet security relies heavily on public-key cryptography, particularly Rivest-Shamir-Adleman (RSA) encryption. This system, developed in 1977, revolutionized secure communication by enabling private information exchange without requiring pre-shared secret keys. RSA's security is based on the mathematical challenge of factoring large numbers into their prime components. For instance, while multiplying two prime numbers is computationally straightforward, the reverse operation—finding the original prime factors of their product—becomes exponentially more difficult as the numbers grow larger.

A. Current Cryptographic Vulnerabilities

Modern RSA implementations typically utilize prime numbers approximately 313 digits in length. Using classical computers and the most efficient known algorithm (the General Number Field Sieve), factoring products of such primes would require approximately 16 million years. However, quantum computers fundamentally alter this security landscape through their unique computational capabilities.

Quantum Computing Advantage - Unlike classical bits that exist in either 0 or 1 state, quantum bits (qubits) can exist in a superposition of both states simultaneously. This property enables quantum computers to perform parallel computations that are impossible for classical computers. With n qubits, a quantum computer can represent 2^n states simultaneously, allowing for the processing of an exponentially large number of possibilities. For example: 2 qubits: 4 simultaneous states, 3 qubits: 8 simultaneous states, 20 qubits: Over 1 million simultaneous states, 300 qubits: More states than particles in the observable universe

B. The Quantum Threat

In 1994, Peter Shor demonstrated that quantum computers could efficiently factor large numbers using quantum Fourier transforms, essentially breaking RSA encryption. This capability has led to a concerning security threat known as "Store Now, Decrypt Later" (SNDL), where adversaries collect and store currently encrypted data, anticipating future quantum capabilities to decrypt it.

The quantum approach to factoring uses Shor's algorithm, which can be simplified into these key steps:

1. Creation of quantum superposition
2. Application of quantum Fourier transform
3. Period finding in modular arithmetic
4. Classical post-processing to derive prime factors

C. Post-Quantum Cryptography Solutions

To address these vulnerabilities, the National Institute of Standards and Technology (NIST) initiated a standardization process for post-quantum cryptographic algorithms. Among the selected solutions, lattice-based cryptography has emerged as a promising approach, particularly through the CRYSTALS-Kyber and CRYSTALS-Dilithium algorithms.

Lattice-Based Cryptography Fundamentals - Lattice-based cryptography operates on mathematical structures called lattices, which are sets of points in n-dimensional space with a periodic structure. The security of these systems relies on two computationally hard problems:

- The Shortest Vector Problem (SVP): Finding the shortest non-zero vector in a lattice
- The Closest Vector Problem (CVP): Finding the lattice point closest to a given point in space

These problems become exponentially harder as the dimension of the lattice increases, making them resistant to both classical and quantum attacks. In practical implementations:

- CRYSTALS-Kyber uses Module Learning With Errors (MLWE) for key encapsulation
- CRYSTALS-Dilithium employs Module Learning With Rounding (MLWR) for digital signatures

D. Paper Contribution

This paper presents 'pqc', a JavaScript implementation of NIST's post-quantum cryptographic standards. Our implementation includes:

- ML-KEM (based on CRYSTALS-Kyber) for key encapsulation
- ML-DSA (based on CRYSTALS-Dilithium) for digital signatures
- SLH-DSA (based on SPHINCS+) for hash-based signatures

II. Related Work

The standardization of post-quantum cryptographic algorithms by NIST represents a crucial milestone in cryptographic history. The three standardized algorithms were selected after rigorous evaluation of their security, performance, and implementation characteristics. This section examines the original algorithms and their evolution into NIST standards.

A. CRYSTALS-Kyber and ML-KEM (FIPS 203)

CRYSTALS-Kyber, the foundation for ML-KEM, is a lattice-based key encapsulation mechanism based on the module learning with errors (MLWE) problem. Originally submitted to the NIST Post-Quantum Cryptography standardization process in 2017, Kyber has undergone several refinements to enhance its security and performance characteristics. Key characteristics of CRYSTALS-Kyber include:

- Modular lattice structure for efficient implementation
- Configurable security levels (Kyber512, Kyber768, Kyber1024)
- Compact public keys and ciphertexts
- IND-CCA2 security in the quantum random oracle model

The standardization as FIPS 203 (ML-KEM) introduced several modifications:

- Refined parameter sets for enhanced security margins
- Standardized encoding formats
- Additional implementation requirements for consistent security across platforms
- Defined test vectors for implementation verification

B. CRYSTALS-Dilithium and ML-DSA (FIPS 204)

CRYSTALS-Dilithium, which forms the basis for ML-DSA, is a lattice-based digital signature scheme. Like Kyber, it utilizes module lattices but employs the "Fiat-Shamir with Aborts" transformation to create efficient signatures.

Notable features of CRYSTALS-Dilithium include:

- Strong theoretical security foundations
- Efficient signature generation and verification
- Relatively small signature and key sizes compared to other post-quantum signature schemes
- EUF-CMA security in the quantum random oracle model

C. SPHINCS+ and SLH-DSA (FIPS 205)

SPHINCS+ represents a different approach to post-quantum security, utilizing hash-based signatures rather than lattice-based mathematics. This conservative approach builds upon decades of research in hash functions and stateless signatures.

Key aspects of SPHINCS+ include:

- Security based solely on the properties of cryptographic hash functions
- Stateless design eliminating the need for signature state management
- Multiple parameter sets trading between signature size and speed
- Minimal security assumptions compared to lattice-based approaches

The standardization as FIPS 205 (SLH-DSA) introduced:

- Specific parameter selections for different security levels
- Standardized instantiations with SHA-2 and SHAKE
- Formal specification of encoding and validation procedures
- Implementation considerations for consistent security

III. Standards & Algorithms

A. FIPS 203: ML-KEM Standard

FIPS 203 standardizes ML-KEM as a key-encapsulation mechanism enabling two parties to establish a shared secret key over a public channel. The scheme's security is based on the computational difficulty of the Module Learning with Errors (MLWE) problem, making it resistant to both classical and quantum attacks.

ML-KEM operates in a polynomial ring

$$Rq := Zq [X] / (X^n + 1)$$

The scheme employs an efficient Number-Theoretic Transform (NTT) that maps polynomials from Rq to their NTT representation in an isomorphic ring Tq. This transformation enables significantly faster polynomial multiplication through coordinate-wise operations in Tq, making it an integral part of ML-KEM rather than just an optimization.

The standard defines three parameter sets offering different security-performance trade-offs. ML-KEM-512 (Category 1), ML-KEM-768 (Category 3), and ML-KEM-1024 (Category 5) differ in their matrix dimension parameter k (2, 3, and 4 respectively) and other internal parameters that affect noise sampling. All variants demonstrate extremely low decapsulation failure rates ($< 2^{-138.8}$), ensuring reliability in practical applications. The Australian Signals Directorate (ASD) mandates the use of Category 5 (ML-KEM-1024) after 2030.

Implementation requirements specify that random bytes must be generated using an approved Random Bit Generator (RBG) with security strength of at least 128 bits for ML-KEM-512, 192 bits for ML-KEM-768, and 256 bits for ML-KEM-1024. The standard prohibits floating-point arithmetic to prevent rounding errors and mandates secure destruction of intermediate values. The matrix A generated during key generation can be stored for later operations, as it's easily

computed from the public encapsulation key and requires no special protections.

B. FIPS 204: ML-DSA Standard

FIPS 204 standardizes ML-DSA for digital signatures that remain secure against quantum adversaries. Based on the Module Learning With Errors (MLWE) problem and using a nonstandard variant called SelfTargetMSIS, ML-DSA provides strong unforgeability under chosen message attacks (SUF-CMA). Similar to ML-KEM the algorithm implements optimization techniques including Number Theoretic Transform (NTT) for efficient polynomial arithmetic.

The standard offers three parameter sets with increasing security levels: ML-DSA-44 (Category 2) with 1,312B public key and 2,420B signatures, ML-DSA-65 (Category 3) with 1,952B public key and 3,309B signatures, and ML-DSA-87 (Category 5) with 2,592B public key and 4,627B signatures. The matrix dimensions (k,ℓ) for these sets are (4,4), (6,5), and (8,7) respectively, offering different security-performance trade-offs.

The specification includes both deterministic and hedged variants of signing, with the hedged variant being the default to mitigate side-channel attacks. Implementation requirements mandate random bit generation with security strength of at least 192 bits for ML-DSA-65 and 256 bits for ML-DSA-87, while ML-DSA-44 requires a minimum of 128 bits. The standard emphasizes protection against timing attacks by prohibiting floating-point arithmetic and mandates proper destruction of intermediate values to prevent potential security compromises.

C. FIPS 205: SLH-DSA Standard

FIPS 205 standardizes SLH-DSA, a stateless hash-based signature scheme that offers a fundamentally different approach to post-quantum security compared to the lattice-based ML-KEM and ML-DSA. The algorithm combines a few-time signature scheme (FORS) for signing message digests with a multi-time signature scheme (XMSS) using Winternitz One-Time Signature Plus (WOTS+) as components, making it resistant to quantum attacks through the computational difficulty of finding preimages for hash functions.

The standard specifies twelve parameter sets with six sets of core parameters, each instantiated with either SHA-2 or SHAKE hash functions. These include variants like SLH-DSA-SHA2-128s/f, SLH-DSA-SHA2-192s/f, and SLH-DSA-SHA2-256s/f, where 's' indicates smaller signatures with slower operation and 'f' denotes faster operation with larger signatures. For instance, SHA2-192f offers Category 3 security with 48-byte public keys and 35,664-byte signatures, demonstrating the characteristic trade-off of hash-based signatures between signature size and performance.

Implementation requirements mandate the use of approved random bit generators with security strength of at least 8n bits, where n is 16, 24, or 32 depending on the parameter set. The standard prohibits floating-point arithmetic to prevent rounding errors and emphasizes the importance of proper destruction of sensitive intermediate data. Unlike its stateful counterparts, SLH-DSA eliminates the need for state management, making it more practical for typical digital signature applications while maintaining strong security properties.

IV. 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.

V. 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.

VI. Future Work

As post-quantum cryptography continues to evolve, several directions for future work emerge. The NIST Round 4 submissions, including Bit Flipping Key Encapsulation (BIKE), Classic McEliece (merged with NTS-KEM), and Hamming Quasi-Cyclic (HQC), present opportunities for

expanding our JavaScript library. These algorithms offer different trade-offs in terms of key sizes, performance, and security assumptions, as analyzed in recent research. 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, offering a smooth transition path for existing applications. Performance optimizations through WebAssembly could provide significant speed improvements while maintaining the accessibility of a JavaScript API. Additionally, as side-channel attacks on post-quantum implementations continue to emerge, implementing countermeasures against these vulnerabilities remains an important area for future work.

As NIST's standardization process continues with Round 4, staying aligned with emerging requirements and potential parameter adjustments will be crucial for maintaining the library's practical utility in post-quantum cryptography.

VII. Conclusion

This research presents the development and implementation of a comprehensive JavaScript library for post-quantum cryptography, conforming to the National Institute of Standards and Technology's standardized algorithms. The implementation encompasses three distinct cryptographic primitives: Module-Lattice-based Key Encapsulation Mechanism (ML-KEM) as specified in FIPS 203, Module-Lattice-based Digital Signature Algorithm (ML-DSA) as defined in FIPS 204, and Stateless Hash-based Digital Signature Algorithm (SLH-DSA) as outlined in FIPS 205.

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.

This work demonstrates the feasibility of implementing standardized post-quantum cryptographic algorithms in pure JavaScript, contributing to the transition toward quantum-resistant cryptographic systems. The implementation serves as a foundation for further research and development in post-quantum cryptography for web applications, while providing immediate practical utility for systems requiring quantum resistance.

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