IMPLEMENTATION AND DEPLOYMENT OF POST-QUANTUM CRYPTOGRAPHY

Gabriel Chênevert[1]
[1] ICL, Junia, Université Catholique de Lille, LITL, F-59000 Lille, France.

ABSTRACT: This short expository note aims to share some of the insight gained by implementing ”from scratch”, in C++, the ML-KEM and ML-DSA quantum resistant cryptographic primitives. Proper understanding of the inner workings of these recently standardized algorithms allows one to produce test vectors to verify compliance of any new implementation, as well as provide small (unsafe) parameter values that can be used for pedagogical purposes.

KEYWORDS: *post-quantum cryptography, module learning with errors, number-theoretic Fourier transform, ML-KEM, ML-DSA*

1. ML-BASED POST-QUANTUM PRIMITIVES

In 2024, the US National Institute of Standards and Technology (NIST) published, after a 7-year-long international selection process, standards for three quantum-resistant algorithms: ML-KEM [2] (formerly known as CRISTALS-Kyber), ML-DSA [3] (CRISTALS-Dilithium) and SLH-DSA [4] (SPHINCS$^+$). The first two of these, based on the *module-learning with errors* (M-LWE) problem [1], share many conceptual ideas and certain practical considerations. A blog post aimed at computer engineers [6] prompted this author, aided by a team of masters-level students, to come up with their own implementations of the then-drafts for the would-be standards in order to be able to provide feedback and compare them to reference implementations. This endeavour fits into a larger goal of being able to assess both the performance and total cost of deploying quantum-resistant primitives, in terms of development effort, scaling of infrastructures, and the actual migration to new primitives.

The basic *learning with errors* (LWE) problem [5] is a problem in linear algebra that can be used as a basis for cryptographical primitives. Given a vector $\mathbf{x}$ and matrix $\mathbf{A}$, it is easy to compute, via matrix multiplication, $\mathbf{y} = \mathbf{A}\,\mathbf{x}$. The inverse process, recovering $\mathbf{x}$ from $\mathbf{y}$ knowing $\mathbf{A}$, it just solving a compatible system of linear equations and can be readily performed using Gaussian elimination. However, if noise is added:

$$\mathbf{y} = \mathbf{A}\,\mathbf{x} + \mathbf{e},$$

where $\mathbf{e}$ is ”small” in some suitable sense, then it becomes considerably more difficult to solve the noisy system of linear equations

$$\mathbf{y} \approx \mathbf{A}\,\mathbf{x}.$$

This asymmetrical problem can be used as the basic for a public-key encryption primitive as follows.

Private key: a secret (column) vector $\mathbf{x}$.

Public key: a matrix $\mathbf{A}$ along with a vector $\mathbf{y}$ such that $\mathbf{y} \approx \mathbf{A}\,\mathbf{x}$.

Encryption of a message m under public key $(\mathbf{A},\mathbf{y})$: choose a random (row) vector $\mathbf{u}$ and compute

$$c = m + \mathbf{u}\cdot\mathbf{y} \qquad \text{as well as} \qquad \mathbf{v} \approx \mathbf{u}\,\mathbf{A}.$$

Decryption of $(c,\mathbf{v})$ using private key $\mathbf{x}$: compute

$$\mathbf{u}\cdot\mathbf{y} \approx \mathbf{u}\,(\mathbf{A}\,\mathbf{x}) = (\mathbf{u}\,\mathbf{A})\,\mathbf{x} \approx \mathbf{v}\cdot\mathbf{x},$$

substract it from c, and remove extra noise to recover m.

ML-KEM is a key establishment mechanism that applies the Fujisaki-Okamoto transform to a simple public-key encryption (PKE) scheme as above, while ML-DSA applies a version of the Fiat-Shamir transform to the asymmetrical problem in order to obtain a Schnorr-like signature scheme. Both use the ”module” flavor of the LWE problem, in which noise is added as low-order bits of coefficients, and components are taken in the polynomial ring that is described in the next section.

2. THE ARITHMETIC OF CYCLIC POLYNOMIALS

Given a ring R of coefficients, consider polynomial expressions

$$f(X) = f_0 + f_1X + f_2X^2 + \cdots$$

with coefficients in R and for which we convene that $X^N = 1$ for some integer power $N > 1$. Just like for modular arithmetic, this allows to work with polynomials of a bounded size even when multiplications are performed.

Elements in the ring $R[X]/(X^N - 1)$ thus constructed can be viewed as vectors in R^N with components arranged in a circle, where :

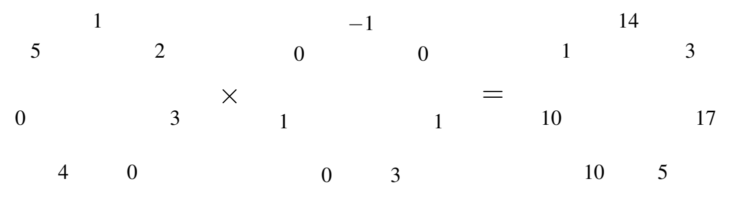
- addition is performed component-wise;
- multiplication by X is a circular permutation (rotation by $\frac{1}{N}^{\text{th}}$ of a turn);
- multiplication in general corresponds to ”circular convolution”:

$$\left(\sum_i f_iX^i\right)\cdot\left(\sum_j g_jX^j\right) = \sum_k h_kX^k \qquad \text{with} \qquad h_k = \sum_{i+j\equiv k} f_i\,g_j,$$

  the sum being taken of pairs of indices (i,j) such that $i+j \equiv k \bmod N$.

For example, we may compute in $\mathbb{Z}[X]/(X^7-1)$ the product $h(X)$ of

$$f(X) = 1 + 2X + 3X^2 + 4X^4 + 5X^6 \qquad \text{and} \qquad g(X) = -1 + X^2 + 3X^3 + X^5 :$$

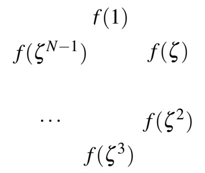


that is,

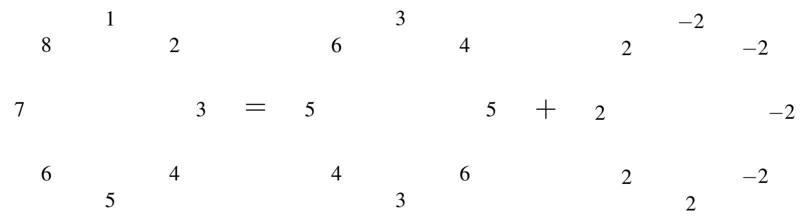
$$h(X) = 14 + 3X + 17X^2 + 5X^3 + 10X^4 + 10X^5 + X^6.$$

The point is that these multiplications can be computed efficiently. A straightforward implementation of the above definition would yield a multiplication algorithm taking $\mathscr{O}(N^2)$ operations, thus considerably slower, when N is large, than pointwise multiplication – which takes only $\mathscr{O}(N)$ operations. However, when N is highly composite (*e.g.*, a power of 2), applying a *fast Fourier transform* (Cooley-Tukey algorithm) to cyclic vectors allows to compute convolutions as pointwise multiplication; the global complexity is dominated by that of the FFT algorithm, $\mathscr{O}(N \log N)$, making circular convolution only marginally slower than pointwise multiplication.

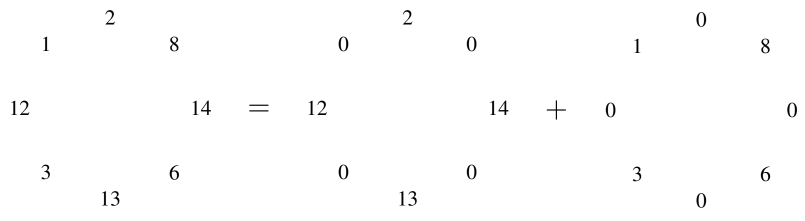
In order to compute the Fourier transform of a cyclic polynomial $f(X) \in R[X]/(X^N - 1)$, a primitive N^{th} root of unity ζ must be chosen in R; the Fourier transform of f can then be thought of as the cyclic vector (in R^N) obtained by evaluating f at the powers of ζ :



Moreover, when $N = 2n$ is even, any cyclic polynomial can be decomposed in to a periodic and antiperiodic part corresponding to the decomposition of the Fourier transforms into even and odd components, obtained by evaluation of the polynomial at even and odd powers of ζ, respectively. For example, with $N = 8$, we have



that correspond, taking Fourier transforms with respect to $\zeta = 2 \bmod 17$, to



This decomposition can be thought of algebraically as the factorization of $R[X]/(X^{2n} - 1)$ into $R[X]/(X^n - 1)$ and $R[X]/(X^n + 1)$ given by the Chinese Remainder Theorem. Both ML-based standards use the latter ring of antiperiodic (or *negacyclic*) polynomials $R[X]/(X^n + 1)$ with $R = \mathbb{Z}/q\mathbb{Z}$ a ring of modular integers. At their core, these algorithms thus specify a triple (q, n, ζ) of constants that are used throughout, where

- q is a prime number used as modulus for the integral coefficients of the polynomials;
- n is the size of the negacyclic polynomials used (typically a power of 2 in order to have access fast multiplication through FFT);
- ζ an ℓ^{th} root of unity in $\mathbf{Z}/q\mathbf{Z}$ (which requires that ℓ divides $q-1$ in order to exist).

Algorithm	n	q	ζ	ℓ
ML-DSA	256	$8380417 = 2^{23} - 2^{13} + 1$	1753	512
ML-KEM	256	$3329 = 2^8 \cdot 13 + 1$	17	256

Table 1: Algebraic parameters for ML-KEM and ML-DSA

Table 1 references the constants used by the ML standards, chosen for the balance they bring between the security level of the primitives and the efficiency of computations involved. We may remark that, in the case of ML-KEM, $\ell = n$ and not $2n$, which complicates matters a bit because a non-split version of the Fourier transform needs to be used, grouping together the factors

$$(X - \sqrt{\zeta}^i)(X + \sqrt{\zeta}^i) = X^2 - \zeta^i$$

in the factorization of $X^{2n} - 1$.

3. COMPARISION WITH CLASSICAL PRIMITIVES

The main asymmetrical cryptographic primitives in use today for signature and key establishment, either based the difficulty of factorization or the discrete logarithm problem (DLP) over the modular integers and elliptic curves, would be vulnerable to an adversary having access to a large enough fault-tolerant quantum computer able to run Shor's algorithm. The main advantage of ML-KEM and ML-DSA over these is that they are oblivious to such attacks; however, there is a price to pay, in terms of both spatial and temporal performances, to achieve this quantum resistance.

For instance, Table 2 references the sizes of the public keys needed in each case to achieve a given level of security.

4. CONCLUDING REMARKS

The main takeaway from this experiment is that both ML-KEM and ML-DSA, as described by the standards, are considerably more subtle to grasp for the average working software developper than modular integer-based classical algorithms (which are relatively well understood by the community) and even elliptic curve-based ones (which still carry an aura of mystery despite being around for almost as long and having seen widespread usage for the last 25 years). In our opinion, this is due in no small part to the fact that the

security level	elliptic curve-based	integer-based	ML-DSA	ML-KEM
128	256	3072	10496	13056
256	512	15360	20736	25344

Table 2: Size (in bits) of the public keys for a given security level

n (non-split)	n (split)	q	ζ	ℓ
2	4	5	2	4
4	8	17	2	8
8	16	17	3	16
16	32	97	3	32

Table 3: Toy parameters that can be used for M-LWE

number-theoretic Fourier transform (NTT) is not used merely as an implementation optimization, but rather embroidered directly into the standards, rendering them somewhat more cumbersome to get a hold on.

Ready availability of test vectors since the publication of the final versions of the standards helps greatly to assess whether an implementation is functionally compliant or not. We suggest that, *for pedagogical purposes*, some implementations may support as ”hazardous material” some smaller (unsafe) parameter choices that would allow people to get a better understanding of the inner workings of these new algorithms, such as those in Table 3. In particular, when speed of execution is not an issue, a modified version of the algorithm might skip altogether the NTT parts and work instead with the slower convolution-style multiplication (for the same functional results).

5. ACKNOWLEDGEMENTS

Many thanks to friend and colleague Pierre Dubois for his precious help with the TikZ figures; as well as to JUNIA ISEN Cybersecurity students Rémi Protin, Aurélien Degain, Hamza Berbache, Enzo Barea Fernandez, Clément Gorse and Benoît Wattinne for their work on our C++ ML-KEM implementation.

References

[1] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. ”(Leveled) Fully Homomorphic Encryption without Bootstrapping.” *ITCS*, ACM (2012): 309–325.

[2] National Institute of Standards and Technology. *Module-Lattice-Based Key Encapsulation Mechanism Standard*, (Department of Commerce, Washington, D.C.), Federal Information Processing Standards Publication (FIPS) NIST FIPS 203 (2024). https://doi.org/10.6028/NIST.FIPS.203.

[3] National Institute of Standards and Technology. *Module-Lattice-Based Digital Signature Standard*, (Department of Commerce, Washington, D.C.), Federal Information Processing Standards Publication (FIPS) NIST FIPS 204 (2024). https://doi.org/10.6028/NIST.FIPS.204.

[4] National Institute of Standards and Technology. *Stateless Hash-Based Digital Signature Standard*, (Department of Commerce, Washington, D.C.), Federal Information Processing Standards Publication (FIPS) NIST FIPS 205 (2024). https: //doi.org/10.6028/NIST.FIPS.205.

[5] O. Regev. ”On Lattices, Learning with Errors, Random Linear Codes, and Cryptography.” *STOC*, ACM (2005): 84–93.

[6] F. Valsorda. ”Enough Polynomials and Linear Algebra to Implement Kyber.” *Cryptography Dispatches* (2023). https://words.filippo.io/dispatches/kyber-math/.