Quantum Safe NIST Candidates

Andrew Nash, 2021; Optum Technologies, Letterkenny

Kyber is one of the 4 NIST finalist cryptography algorithms which (should) offer post-quantum security. Of the 4 Key Encapsulation finalists, this algorithm is most likely to see widespread adoption, so will be the subject of most focus here, with justification for this in the latter half of the document.

Kyber’s security, as do 2 other KEM finalists (NTRU, Saber) depends on solving problems involving *lattices.*

# lattice based cryptography

Lattice based algorithms typically have very small key sizes < 1.5KB, often on the order of hundreds of bytes, and faster than RSA to encrypt and decrypt. There is a decent, simple explanation of what a lattice is at <https://medium.com/cryptoblog/what-is-lattice-based-cryptography-why-should-you-care-dbf9957ab717> . NTRU, Saber and Kyber are based on a somewhat problem to SVP mentioned in the article – Learning With Errors/Rounding (which is related to it, however).

## Kyber Algorithm

An explanation of the Kyber algorithm for key encapsulation (KEM), and how it relates to Module Learning With Errors is given below. Note that as a KEM algorithm the implementation below assumes a fixed key size of 32 bits that acts as the message to be encrypted. Multiple iterations of the algorithm are performed when longer messages are involved.

### Key Generation

**A** - a k by k matrix, generated from the hash of a seed sampled from a uniform distribution. A random A is generated by the server, which the client can recreate from the public key

**s** – a length k vector, with elements sampled from a centred binomial distribution

**e** – a length k vector, with elements sampled from a centred binomial distribution

This is somewhat a simplification, as Kyber does not work with matrices and vectors of integers, but polynomials in a specific quotient vector space. We will ignore this temporarily, without losing the core ideas of the algorithm.

Let:

**t**, combined with the seed used to generate **A** form the public key.

**s** acts as the private key

### Encryption

1. The server generates **A** from the public key, as discussed above.
2. A new vector **r** of length k is sampled from a centred binomial distribution
3. **e1** , **e2** are similarly sampled (*from a potentially different centred binomial distribution*)

Where u,v form the ciphertext. Note that the full algorithm also performs compression and decompression on the ciphertext, which reduces ciphertext sizes, and also provides error tolerance if implemented correctly.

### Decryption

Decryption is (without the encoding, decompression steps) simply

### Breaking LWE Decryption

It can be proven that solving these equations for unknowns reduces to MLWE, as follows (unrigorously, any errors in justification here are my own)

Consider

An attacker will be able to derive **A** and **t** from the public key, but **e** is an unknown. Were it not there, the attacker would be able to relatively trivially solve for **s** with a method such as Gaussian elimination. It is conjectured that versions of this LWE problem are NP-hard for both quantum and classical systems, but no proof has yet to be demonstrated. There is therefore a risk that over the next number of years and decades, a proof will be produced that will lead to practical attacks on many LWE-based algorithms.

### Matrix and Vector Elements

As I mentioned earlier, it was heavily implied above that vector and matrix elements were integers, or at the very least Real. For Kyber however (and most all other lattice based cryptography algorithms), it is more secure and effective to work with elements in the ring , where n is a power of two. Some lattice based algorithms use different quotients in the above equation, but this seems to be becoming less and less common, with the trend being for algorithms to evolve towards using the same ring.

More transparently, this ring corresponds to the set of all polynomials of degree less than n, with integer coefficients modulo q.

This means that for every vector addition and matrix multiplication we perform above, the operations are between polynomials. If computed naively, these would take O(n) operations for each addition of two individual polynomials and O(n^2) operations for *each individual multiplication* between polynomials of degree n*.* However, by applying a version of the Discrete Fourier Transform known as the Number Theoretic Transform, we can reduce this to the order of O(n log n) operations. Note that this is dependent on the choice of n – (it is chosen as a power of two > 2, due to some characteristics of roots of unity in power-of-two rings, but this is not important to understand). Further, the NTT of two polynomials can be computed without requiring extra memory and can be implemented in a heavily optimized manner that brings the number of clock cycles used to compute an NTT in almost k cycles.

The underlying principles of the NTT are similar (albeit taken to a slightly higher level of complexity) to those of using a FFT to perform ordinary polynomial multiplication. There is an explanation (although quite dense and not very well structured) here for using a FFT to multiply polynomials efficiently

https://medium.com/@aiswaryamathur/understanding-fast-fourier-transform-from-scratch-to-solve-polynomial-multiplication-8018d511162f

## Points of Note About Kyber

### Utilising hardware optimisation

When benchmarking implementations of Kyber, it is worth bearing in mind that it has been noted in the official specification for Kyber that the majority of the execution time for Kyber encryption, decryption and key generation (given properly implemented NTT and inverse NTT functions) is spent computing hashes, using algorithms including versions of SHA3, SHA2, SHAKE and AES. If any hardware accelerated implementations of these algorithms are available on a system, it is highly beneficial to use an implementation of Kyber that capitalizes on this.

### No Guarantee That a Message is Decryptable

The specification of Kyber suggests a number of sets of parameters that deliver estimated increasing levels of classical and quantum security. Each of these has been shown to have no guarantee that a message encrypted with an arbitrary key can be decrypted, but the probability of this happening is between 2-139 and 2-174, a quantity so insignificant that no errors can be expected to occur even with universal adoption, for the remainder of the lifespan of the universe. Thus, its chances are dwarfed by those other potential system failures and can be discounted. This is also the case in other lattice based methods, including both Saber and NTRU.

### No Quantum or Classical Security Proof

The MLWE problem has not yet been proven to be NP-hard. It is worth noting that rings/vector spaces such as the ring of polynomials used in Kyber have an extremely high amount of structure – the possibility of this being exploited to arrive at a polynomial algorithm is very real. As a counterpoint, the exact sort of ring used by Kyber, with q as a prime number and the quotient , as a cyclotomic polynomial have been heavily researched in terms of their properties for hundreds of years. Nonetheless, it would be extremely imprudent to deploy Kyber as a sole algorithm used to encrypt sensitive data. On the other hand, it is very desirable to being securing communications against quantum attacks in the very near future – while large scale error corrected quantum computing is still likely a few decades a way at least, this horizon is less than the confidentiality of much data being transmitted today. This leaves an ever-growing incentive for attackers to intercept, and store sensitive data that will retain value over many years. This also applies (arguable to an even greater extent) to NTRU and Saber.

The best compromise that addresses this issue is to implement a hybrid approach of standard, provably classically secure, Elliptic Curve or RSA encryption in conjunction with Kyber. In this configuration, the security of the encrypted data can only be compromised with both access to a large-scale error corrected quantum computer, as well as a polynomial time solution to the MLWE problem – a much less likely prospect than both happening individually.

## Saber

Saber is similar to Kyber in many respects, but the most pertinent difference is that its security is based on Learning With Rounding, rather than learning with errors – instead of adding an error term that makes the secret key difficult to solve for, it rounds the elements of the resulting vector. Note that rounding has more algebraic structure than addition of well-distributed errors, which could be a (although there is no evidence as yet of such) vulnerability. There is in fact some rounding of the polynomial coefficients in Kyber, which is not relied heavily on in its theorized security, but is beneficial in increasing its hardness slightly – adding about 8 bits of security. This does give the impression of Kyber being more secure than Saber by having both noise and rounding elements that need to be solved by an attacker (by my estimation). There is some evidence that key generation, encryption and decryption are slightly faster than all other methods mentioned.

## NTRU

NTRU is also very similar to Kyber in many respects, but has some disadvantages over it and, to the best of my knowledge, no discernable advantages. Firstly, it involves polynomial division rather than multiplication, which is less efficient to compute. Further, due to the choice of underlying ring, it is often not possible to use the NTT method to be used to speed up these divisions. Secondly, the lattice in which NTRU operates has some algebraic structure that have been shown to be potential weaknesses to attack.

Classic McEliece

Classic McEliece is being considered by NIST as an alternative candidate, mainly should the underlying problems on lattices above are shown to be solvable in polynomial time. It was introduced in 1978 and has been thoroughly researched since then – we can thus be somewhat more confident in its security than the lesser-researched and unproven lattice methods. The underlying principle that creates its security involved decoding certain Error correcting Codes. It has the major drawback of having extremely large keys, from roughly 200KB – 1.5MB, although ciphertexts can be quite short (less than 250 bytes) and encryption and decryption are still faster than RSA (as are all of the above algorithms). This all means that this is not really feasibly in most scenarios, but there may be some very limited instances where the larger key sizes are acceptable, and the higher confidence in its proof of security over lattice-based methods may be much more useful.

# Summary of key sizes and security (KEM)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Private Key Size (bytes)  [compressed] | Public Key Size (bytes) | Ciphertext (bytes) for 32 bit key | Estimated/Targeted Bits of Security |
| Kyber-512 | 1632 | 80 | 768 | 128 |
| Kyber-768 | 2400 | 1184 | 1088 | 192 |
| Kyber-1024 | 3168 | 1568 | 1568 | 256 |
| NTRUhps2048509 | 935 | 699 | 699 | 128 |
| NTRUhps2048677 | 1234 | 930 | 931 | 192 |
| NTRUhps4096821 | 1590 | 1230 | 1230 | 192 |
| NTRUhrss701 | 1452 | 1138 | 1138 | 256 |
| LightSaber | 1568 [992] | 672 | 736 | 118/107 (classical/quantum) |
| Saber | 2304 [1440] | 992 | 1088 | 189/172 |
| FireSaber | 3040 [1760] | 1312 | 1472 | 260/236 |
| mceliece348864 | 6492 | 261120 | 128 | 128 |
| mceliece348864f | 6492 | 261120 | 128 | 192 |
| mceliece460896 | 13608 | 524160 | 188 | 192 |
| mceliece460896f | 13608 | 524160 | 188 | 256 |
| mceliece6688128 | 13932 | 1044992 | 240 | 256 |
| mceliece6688128f | 13932 | 1044992 | 240 | 256 |
| mceliece6960119 | 13948 | 1047319 | 226 | 256 |
| mceliece6960119f | 13948 | 1047319 | 226 | 256 |
| mceliece8192128 | 14120 | 1357824 | 240 | 256 |
| mceliece8192128f | 14120 | 1357824 | 240 | 256 |

# Performance comparison

It is quite difficult at present to benchmark the performance differences between algorithms at present, for the main reason that Saber, NTRU and Kyber all make use of hashing algorithms – namely SHA2 and SHA3 at various security levels. Optimised implementations of all 3 algorithms can still differ in relative performance on different systems, if the test is not adjusted to account for use of older, well established hashing algorithms that are hardware accelerated. Further, while very, very few systems support hardware accelerated SHA3 at present, AMD for example have begun the process of including support in newer hardware, which will allow these algorithms to gain the benefit from hardware acceleration, without having to deviate from their base specified hashing algorithms.

Based on the specification documents, this ranking below should provide a rough indication of the expected performance of different algorithms in different processes

## Key Generation – lower ranking is faster/better

LightSaber < Kyber512 << NTRUhps2048509

Saber < Kyber768 << NTRUhps2048677 < NTRUhps4096821

FireSaber ~= (almost equal, slightly less) Kyber1024 <<< NTRUhrss701 (far more)

## Encryption

Kyber512 < LightSaber < NTRUhps2048509

Kyber768 < Saber << NTRUhps4096821

NTRUhrss701 << Kyber1024 < FireSaber

## Decryption

LightSaber < Kyber512 < NTRUhps2048509

Kyber768 < Saber << NTRUhps4096821

NTRUhrss701 << FireSaber < Kyber1024

Note that no versions of McEliece are mentioned above. These all (even the lowest security versions) have key generation times orders of magnitude lower than all the versions of the other algorithms. Encryption and decryption times seem roughly on par with their equivalent-security counterparts of other algorithms, which is still less than that of RSA.

Information above is based largely on:

<https://github.com/mupq/pqm4/blob/master/benchmarks.md>

<https://openquantumsafe.org/benchmarking/visualization/speed_kem.html>

# Summary/Conclusion

It is likely that Kyber will be adopted by many entities of the next number of years in a hybrid algorithm alongside conventional RSA or Elliptic Curve algorithms, due to its tradeoff between speed and security – the only other finalist algorithm consistently outperforming it looks to be Saber (and even then, by a very acceptable small margin). Saber looks to potentially be the more likely of the two to be broken in polynomial time, with Kyber incorporating a (very slightly) wider range of and more sophisticated elements in its underlying algorithm.