Lattice-based Cryptography : A Solution to Post-Quantum Cryptography Standardization

April 22, 2024

Abstract

The NIST (National Institute of Standards and Technology) is an agency of the United States, responsible for the development of standards and guidelines to protect information systems. It has published a report on post-quantum cryptography and initiated a process to solicit and standardize quantum-resistant public-key cryptographic algorithms. This article introduces the motivation and goal of the standard process based on the report from NIST, summarizes the structural and methodological construction of the chosen standard draft: Module-Lattice-based Key-Encapsulation Mechanism and makes some comparisons with the initial proposal.

1 Motivation and Goals

In the past thirty years, public key cryptography has become vital for global digital communication, relying on core functionalities like public key encryption, digital signatures, and key exchange, all hinging on challenging number theoretic problems. Research into large-scale quantum computers surged following Peter Shor's 1994 breakthrough, initially met with doubts about scalability due to quantum states' fragility, but later, quantum error correction and threshold theorems demonstrated the feasibility of reliable quantum computations. Recent advancements in hardware and error correction have brought quantum computing closer to reality, yet significant efforts are still needed for large-scale implementation. While quantum algorithms promise exponential speedup for certain problems like physics simulation and number theory, in the realm of cryptography, the security of algorithms such as RSA and DSA, which rely on public key encryption, is compromised by the emergence of large-scale quantum computers. This necessitates the use of larger key sizes or the adoption of alternative cryptographic methods. Additionally, hash functions like SHA-2 and SHA-3 will require larger output sizes to withstand potential attacks from quantum computers.

The goal of standardizing post-quantum cryptography is to develop standards and guidelines for protecting non-national security federal information systems against quantum computers [4]. Many researchers say that in the near future quantum computers will break currently deployed public key cryptography and significantly weaken symmetric key cryptography [6]. Securing a smooth and robust transition from presently employed cryptosystems to their quantum computing-resistant equivalents poses a formidable challenge. The precise timing of the quantum computing era's advent remains uncertain, necessitating ample lead time for fortifying our information security infrastructure against quantum threats. An effective strategy for addressing the challenges posed by quantum computers to data security involves completing the migration to Post-Quantum Cryptography (PQC) algorithms and protocols capable of resisting quantum attacks. This entails replacing all presently utilized cryptographic algorithms, which rely on traditional number theory problems, with their post-quantum cryptographic counterparts. By doing so, the public key cryptographic algorithm is substituted with a version resilient to quantum computing attacks, ensuring not only efficient operation on conventional computers but also effective resistance against quantum threats within feasible timeframes.

Post-quantum cryptography is considered as a solution to resist quantum attacks, but it is important to note that challenges and limitations still exist. The key lengths of

most post-quantum algorithms are longer than those of the algorithms they are intended to replace, posing challenges that require careful consideration by experts when altering protocol schemes. Furthermore, it has not been proven that current post-quantum algorithms can guarantee security against all quantum attacks, which implies the possibility of some quantum attacks being able to compromise post-quantum cryptographic schemes. Nonetheless, post-quantum cryptographic schemes are significant, and the release of post-quantum cryptographic standards will help assess the extent of the threat that quantum computing poses to existing standards.

2 Module-Lattice-based Key-Encapsulation Mechanism Standard

In this section, we will analyze the draft written by NIST: the Module-Lattice-based Key-Encapsulation Mechanism Standard[7].

2.1 The draft structure

To elucidate the purpose of the standard, the document first situates the reader within the topic, providing the reasoning for standardization, contextual information, as well as referencing the original paper and its subsequent adjustments. Subsequently, it offers a glossary of terms, acronyms, and mathematical symbols to aid the reader's comprehension. Following this, the document provides an overview of the proposed standardization: the Key-Encapsulation Mechanism Standard. This overview begins by presenting the fundamental principles, followed by a succinct and informal description of the computational assumptions underlying ML-KEM, and elucidating how the ML-KEM scheme is constructed. Furthermore, it outlines requirements for implementing the algorithms of ML-KEM in a secure manner. Additionally, the document discusses auxiliary algorithms used in ML-KEM.

Subsequently, it explains the K-PKE component Scheme, which constitutes one of the most critical aspects of the ML-KEM Key-Encapsulation Mechanism. In the subsequent section, it details the algorithms for key generation, the encapsulation process, and the decapsulation process. Finally, the standard specifies three parameter sets for ML-KEM, arranged in order of increasing security strength and decreasing performance.

2.2 The Module-Lattice-based Key-Encapsulation Mechanism (ML-KEM)

This section gives an explanation of the ML-KEM scheme. First, we will give an overview of the ML-KEM and then explain more deeply the algorithms.

2.2.1 Overview of the ML-KEM

ML-KEM uses the principle of Key-Encapsulation Mechanism (KEM), which facilitates the establishment of a shared secret key between two parties. A KEM comprises three essential algorithms: KeyGen, Encaps, and Decaps, alongside a collection of parameter sets for security and efficiency trade-offs. In practice, Alice generates an encapsulation key and a decapsulation key using KeyGen. Bob, upon receiving Alice's encapsulation key, utilizes

the Encaps algorithm to generate his copy of the shared secret key (KB) and a ciphertext, which he sends to Alice. Alice, using her decapsulation key and the ciphertext, runs the Decaps algorithm to derive her copy of the shared secret key (KA). However, the assurance of security and randomness in the generated keys is subject to certain assumptions.

The ML-KEM scheme is a key-encapsulation mechanism built upon the computational assumption of Module Learning with Errors (MLWE), which is a generalization of the Learning with Errors (LWE) problem introduced by Regev (see [5] for more details). The ML-KEM construction operates in two steps. Firstly, it constructs a public-key encryption scheme from the MLWE problem. Secondly, it converts this encryption scheme into a key-encapsulation mechanism using the Fujisaki-Okamoto (FO) transform.

2.2.2 Explanation on the Scheme Algorithms

Chapter 5 and 6 of the draft describe the algorithms of the K-PKE component scheme and the ML-KEM scheme respectively, and are organized in the order of key generation, encryption (or encapsulation) and decryption (or decapsulation). Regarding the relationship between the K-PKE component scheme and the ML-KEM scheme, the draft notes that the K-PKE component scheme is used as a subroutine for the ML-KEM scheme, and that K-PKE itself is not sufficiently secure to be used as a separate scheme. In other words, researchers cryptographically transformed K-PKE to obtain a sufficiently secure ML-KEM scheme. The algorithms for these two schemes are explained as follows.

The key generation algorithm of the K-PKE scheme has no inputs and outputs an encryption key and a corresponding decryption key. Ensuring randomness is important to this algorithm. The decryption key is a vector s and the encryption key is a noisy linear equation (A, t) based on the decryption key, where $t = As + e$. The matrix A is pseudo-randomly generated using XOF (eXtendable Output Function), and the seed used by XOF will be appended to the encryption key. The secret s and the noise e are obtained by sampling from centered binomial distribution using PRF (PseudoRandom Function) as parameter. The seeds used by PRF and XOF are generated by a pseudorandom seed extension function whose parameter is still a random number which is required to be generated using the approved RBG (Random Bit Generator), as prescribed in NIST SP 800-90A [3], NIST SP 800-90B [9], and NIST SP 800-90C [2]..

The K-PKE encryption algorithm accepts as input three parameters, the encryption key, the plaintext and the randomness, and outputs the ciphertext. The encryption of the K-PKE encryption algorithm is based on the idea that, without knowing the secret s, an additional noisy linear equation can be generated by choosing a random linear combination on the (A, t) noisy equation, and the plaintext can be encoded into the constant term of the additional linear equation, which can be subsequently decrypted by one in possession of the secret s. When generating the additional noise linear equation, the K-PKE encryption algorithm makes use of a vector and two noises, both of which are sampled from centered binomial distribution using a PRF with the randomness parameter as seed. The K-PKE decryption algorithm accepts the decryption key and the ciphertext as input and outputs the plaintext. The decryption algorithm does not require randomness, it just decrypts the ciphertext using the decryption key. At last, it is worth noting that the K-PKE scheme is consistent with the idea of public key encryption. Therefore, the encryption key can be made public but decryption key and the randomness must be kept secret.

ML- KEM algorithm does not contain too much mathematical details directly, but mainly calls the K-PKE algorithm and makes the transformation. The key generation

algorithm of ML-KEM mainly makes use of the K-PKE key generation algorithm. The encapsulation key of ML-KEM is the encryption key of K-PKE, and the decapsulation key is the decryption key of K-PKE appended with the encapsulation key and its hash. In addition, a random number which will be used in decapsulation process to perform implicit rejection is appended to the the end of the decapsulation key. This random number is required to be generated from the approved RBG.

The ML-KEM encapsulation algorithm accepts the encapsulation key as input and outputs the shared key and ciphertext. The algorithm mainly calls the K-PKE encryption algorithm and carefully constructs the input parameters of this encryption algorithm. Two of the three input parameters of the encryption algorithm are random numbers but are not generated in exactly the same way. The first parameter is given encapsulation key, which is just the K-PKE encryption key, and the second parameter, plaintext, is actually a random number which is still required to be generated by the approved RBG. The third random number parameter is generated by a hash function on the first and second parameter, which can be regenerated in the decapsulation process to perform validation. Shared key to be delivered is generated in the same hashing way. Ultimately, the ML-KEM encapsulation algorithm outputs the shared key and the K-PKE ciphertext.

The ML-KEM decapsulation algorithm accepts the ciphertext and the decapsulation key as input and outputs the shared key. The decapsulation algorithm utilises the K-PKE decryption algorithm and hash function to obtain the shared key and randomness, and also utilises the K-PKE encryption function for additional validation. To be more specific, all the three parameter inputs of the encryption algorithm can be obtained by decryption or regeneration, and then the encryption algorithm is run again and the generated ciphertext is compared with the received ciphertext. If they match, the shared key is accepted, otherwise it is implicitly rejected.

2.3 Differences from the Original Proposals

ML-KEM is derived from the round-three version of the CRYSTALS-KYBER KEM [1], a submission in the NIST post-quantum cryptography standardization project. This section will analyze the differences between ML-KEM and CRYSTALS-KYBER.

2.3.1 Length of Secret Key

In the third-round specification, the shared secret key was handled as a variable-length value, its length contingent on its application-specific usage [1]. However, in FIPS 203, the shared secret key length is set at 256 bits. Consequently, it can be directly employed in applications as a symmetric key, or symmetric keys can be derived from it.

2.3.2 The variant of Fujisaki-Okamoto transform

In the draft, NIST opted for a modified Fujisaki-Okamoto transform by excluding the hash of the ciphertext in the derivation of the shared secret within the Encapsulation algorithm. They also adjusted ML-KEM.Decaps to align with this alteration. This adjustment eliminates the need for hashing the ciphertext during the encapsulation process, leading to a notable speedup. Moreover, it streamlines security proofs without sacrificing integrity. However, this modification introduces a timing side-channel, which manifests as variations in the time taken to execute certain operations or algorithms, potentially disclosing explicit rejection. Consequently, further analysis may be necessary to ensure comprehensive security assurance.[8]

2.3.3 The Hash Operation to the Randomness m

The initial randomness m in the `ML-KEM.Encaps` algorithm was first hashed before being used in the third-round of specification [1]. But in FIPS 203, the hash operation to the randomness m is removed due to this standard requires the use of NIST-approved randomness generation. The randomness m, a fresh string of random bytes, muat be generated using an approbed RBG specified in NIST SP 800-90A [3], NIST SP 800-90B [9], and NIST SP 800-90C [2]. Standardization of randomness generation helps improve post-quantum cryptography system reliability, security and portability.

2.3.4 Explicit Input Validation Steps

FIPS 203 introduces explicit input validation steps that were not included in the third round of specification [1]. Receiving and handling ML-KEM public keys or ciphertexts that are not properly formed must follow the input validation rules set out in FIPS 203. Specifically, when receiving an ML-KEM public key for encapsulation, they must check its type and ensure that it follows the modulus rules as described in Section 6.2 of FIPS 203 (ML-KEM Encapsulation). If the public key doesn't meet these requirements, it should be rejected. Similarly, when receiving an ML-KEM ciphertext for decapsulation, they should check its type and ensure that it's valid according to the rules in Section 6.3 of FIPS203 (ML-KEM Decapsulation). If the ciphertext or the key isn't formed correctly, it should also be rejected. These checks can be done separately before the encapsulation or decapsulation steps or as part of them. Input validation guarantees that all inputs to K-PKE algorithms, when invoked as subroutines, will be valid, and ensures that `ML-KEM.Encaps` and `ML-KEM.Decaps` are only executed on validated inputs.

References

[1] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-kyber algorithm specifications and supporting documentation. 2021.

[2] Elaine B Barker and John Michael Kelsey. *Recommendation for random bit generator (RBG) constructions*. US Department of Commerce, National Institute of Standards and Technology â¦, 2012.

[3] Elaine B Barker, John Michael Kelsey, et al. *Recommendation for random number generation using deterministic random bit generators (revised)*. US Department of Commerce, Technology Administration, National Institute of â¦, 2007.

[4] Lily Chen, Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray A Perlner, and Daniel Smith-Tone. *Report on post-quantum cryptography*, volume 12. US Department of Commerce, National Institute of Standards and Technology â¦, 2016.

[5] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, page 35, 2015.

[6] Michele Mosca. Cybersecurity in an era with quantum computers: will we be ready? Cryptology ePrint Archive, Paper 2015/1075, 2015. `https://eprint.iacr.org/2015/1075`.

[7] National Institute of Standards and Technology. Module-lattice-based key- encapsulation mechanism standard. `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.ipd.pdf`, 2023.

[8] CRYSTALS-Kyber submission team. Discussion about kyberâs tweaked fo transform. `https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/WFRDl8DqYQ4`, 2023.

[9] Meltem Sönmez Turan, Elaine Barker, John Kelsey, Kerry A McKay, Mary L Baish, Mike Boyle, et al. Recommendation for the entropy sources used for random bit generation. *NIST Special Publication*, 800(90B):102, 2018.