



Cryptographie post-quantique

ML-KEM et ML-DSA

Réalisé par :

- Louis Landouzy
- Hamza Berbaché
- Benoît Wattinne
- Clément Gorse
- Enzo Barea
- Aurélien Degain
- Rémi Protin

Supervisé par :

- Gabriel Chênevert

Table des matières

I.	Introduction	3
II.	ML-KEM	4
A.	Problème MLWE	4
i.	Introduction.....	4
ii.	Problème sur les réseaux	4
iii.	LWE.....	4
B.	Number-theoretic Transform (NTT)	5
C.	Création d'un Schéma de chiffrement à clé publique (PKE).....	6
i.	Génération de clé.....	6
ii.	Chiffrement	7
iii.	Déchiffrement	8
D.	Conversion en Mécanisme d'encapsulation de clé (KEM)	8
i.	Transformation FO	8
ii.	Génération de clé.....	8
iii.	Encapsulation.....	9
iv.	Décapsulation	9
III.	ML-DSA	10
IV.	Projection.....	11
	Bibliographie.....	12

I. Introduction

La cryptographie post-quantique émerge comme un pilier essentiel de la sécurité face à la menace grandissante des ordinateurs quantiques, capables de compromettre des protocoles classiques. L'algorithme de Shor est capable de factoriser un entier naturel N en un temps $O((\log N)^3)$, mettant ainsi en péril les systèmes de cryptographie à clé publique comme Diffie-Hellman ou RSA. Dans cette évolution majeure, le NIST propose de nouveaux protocoles en phase de validation, parmi lesquels se distinguent ML-KEM pour l'échange de clés privées et ML-DSA pour la signature électronique.

Ce rapport vise à mettre en lumière les mécanismes et les principes sous-jacents de ces primitives cryptographiques destinées à remplacer, à terme, des protocoles établis, mais vulnérables.

Nous aborderons dans un premier temps le détail technique et fonctionnel de ML-KEM, en établissant d'abord les fondements du problème sur lequel il s'appuie. Puis nous feront un parallèle avec ML-DSA pour le décrire à son tour de manière plus succincte. Enfin, nous terminerons par une projection pour la suite du projet, envisageant l'implémentation du protocole en explorant toutes les possibilités, toujours de manière didactique.

En parallèle de cette exploration technique, ce projet a été mené en utilisant des méthodes de gestion agile. L'utilisation de l'outil de gestion de projet Trello (accès disponible dans la bibliographie) a permis de structurer et de suivre les étapes clé du processus. Des comptes-rendus réguliers ont été établis à chaque réunion, assurant une traçabilité et une gestion efficace des tâches. Chaque contributeur s'est vu attribuer une tâche suite à des discussions visant à définir la direction à prendre pour mener à bien le projet. Notre découpage méthodique, que l'on retrouve notamment dans le plan, a été un élément clé pour mieux appréhender les interactions entre les différents éléments identifiés en amont, et ce, tout au long du projet.

Nous souhaitons exprimer notre gratitude envers M. Chênevert, notre enseignant et responsable de ce projet, pour son précieux accompagnement et ses conseils éclairés tout au long de cette étude. Sa contribution a été fondamentale pour faire avancer notre cheminement et approfondir notre compréhension des problèmes.

II. ML-KEM

A. Problème MLWE

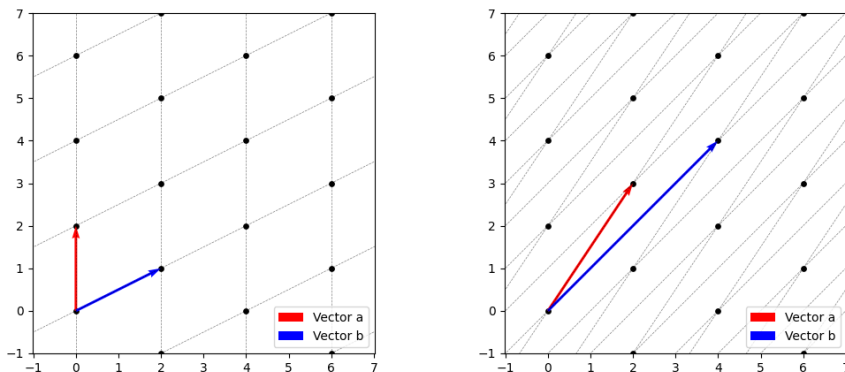
i. Introduction

Le mécanisme d'encapsulation de clé ML-KEM repose sur l'utilisation de problèmes difficiles conjecturés sur des réseaux de points dans $\mathbb{R}^n$. On suppose que la difficulté de résoudre ce problème même dans les conditions les plus défavorables est prouvée. En parallèle, l'apprentissage avec des erreurs (Learning With Errors - LWE) implique la recherche d'une solution à un système linéaire où les coefficients sont perturbés par du bruit. Ces problèmes ont des propriétés mathématiques spécifiques qui les rendent difficiles à résoudre, même pour des ordinateurs quantiques hypothétiques.

ii. Problème sur les réseaux

La sécurité repose sur la difficulté de résoudre le problème du plus court vecteur (Shortest Vector Problem - SVP).

Étant donné un réseau généralement représenté par une base, le SVP consiste à trouver le vecteur non nul le plus court dans le réseau soit le point le plus proche de zéro. Le problème peut être défini par rapport à n'importe quelle norme, mais la norme euclidienne est la plus courante.



Soit un même réseau avec deux différentes bases, à gauche une base plus facile à utiliser qu'à droite. Plus la base est orthogonale, plus les vecteurs sont courts et communiquent des informations sur le réseau.

Ce problème est tellement dur à résoudre qu'on préfère l'approximer avec GAPSVP (Generalized Approximate Shortest Vector Problem). Cette version du problème introduit deux variables de plus, d et γ . Cela permet de trier les réponses possibles en fonction de si le vecteur le plus court est inférieur à d ou s'il est plus grand que γd .

Le SVP est un problème NP-difficile, ce qui signifie qu'il est difficile de vérifier rapidement si une solution donnée est correcte, même si on peut facilement vérifier une solution correcte.

iii. LWE

Le fonctionnement de LWE est basé sur une variante du SVP. La donnée à chiffrer est représentée par un point du réseau. Le point est ensuite légèrement décalé en dehors du réseau. Pour déchiffrer l'objectif n'est pas de trouver le vecteur le plus court, mais plutôt le point du réseau le plus proche.

Voici en pratique comment fonctionne LWE étape par étape :

1. On choisit k qui influe sur le niveau de sécurité $k \in \{2, 3, 4\}$
2. On génère une matrice A de taille $k \times k$, d'entier uniformément aléatoire $A \in (\mathbb{Z})^{k \times k}$
3. On génère un vecteur secret s de taille k , d'entier uniformément aléatoire $s \in (\mathbb{Z})^k$
4. On calcule le produit de la matrice A et le vecteur s : $\langle A \cdot s \rangle = t$
5. On rend publics la matrice A et le résultat t

L'objectif d'un attaquant est de découvrir le vecteur s , mais on se retrouve avec un système à k équation et k inconnus, on peut donc le résoudre efficacement en utilisant la méthode du pivot de Gauss avec une complexité $O(n)$.

$$\begin{array}{c} \xleftrightarrow{k} \\ \begin{array}{c} \updownarrow k \\ \left[\begin{array}{c} A \end{array} \right] \end{array} \cdot \left[\begin{array}{c} s \end{array} \right] + \left[\begin{array}{c} e \end{array} \right] = \left[\begin{array}{c} t \end{array} \right] \end{array}$$

On génère alors le vecteur e de k petits entiers aléatoires répartis sur une loi binomiale $e \in (\mathbb{Z})^k$. On rajoute ce bruit à notre résultat, la résolution devient alors très difficile avec cette légère erreur à chaque ligne.

B. Number-theoretic Transform (NTT)

La NTT (Number-Theoretic Transform) est un protocole de calcul qui est utilisé de nombreuses fois dans ML-KEM, car il permet de réduire la complexité générale des multiplications de polynôme. En effet sans la NTT, la multiplication de grands polynômes prend énormément de temps dans la ML-KEM. Nous passons ainsi d'une complexité de temps de $O(n^2)$ à $O(n \log n)$.

La NTT est similaire à la transformée de Fourier rapide, la seule différence résidant dans le fait que nous utilisons des opérations arithmétiques modulo un nombre premier spécifique et que la NTT a lieu dans un domaine fini. La NTT va mapper un polynôme f dans un certain anneau R_q de la forme :

$$f = f_0 + f_1X + \dots + f_{255}X^{255} \in R_q.$$

vers sa représentation NTT $\hat{f}$ dans un anneau différent T_q . La transformée de Fourier, dans le cas de ML-KEM, est équipée d'une arithmétique modulo $X^n + 1$. Dans le protocole, le choix de l'ensemble de départ est $\mathbb{Z}q^{2n}$. Cela veut dire que nous cherchons une racine 512^{ème} afin de réaliser notre NTT, ce qui n'existe malheureusement pas avec notre valeur de q ($q = 3329$ et $\frac{3328}{512} = 6,5$. Ce nombre n'est pas un entier ce qui prouve qu'aucune racine 512^{ème} n'existe). Afin de résoudre ce problème, nous pouvons écrire que $X^{512} - 1 = (X^{256} - 1)(X^{256} + 1)$. On se retrouve ici avec une fonction antipériodique où les composantes paires sont annulées et nous pouvons ainsi nous permettre de conserver uniquement les termes en $X^{256} + 1$ qui représentent la partie importante de la fonction. Nous nous retrouvons ainsi avec l'arithmétique voulue $(X^n + 1)$. Si la racine 512^{ème} existait, on aurait pu écrire que $X^{256} + 1 = \prod_{imp} (X - w^k)$. Pour contourner ce problème, le polynôme $X^{256} + 1$ se décompose en 128 polynômes de degré 2 modulo q comme suit :

$$X^{256} + 1 = \prod_{k=0}^{127} (X^2 - \zeta^{2\text{BitRev7}(k)+1})$$

Où BitRev7(i) est l'entier représenté par le renversement des bits de la valeur binaire non signée à 7 bits qui correspond à l'entier d'entrée i compris entre 0 et 127. Ainsi, la représentation NTT de $\hat{f} \in T_q$. d'un polynôme $f \in R_q$. est le vecteur suivant :

$$\hat{f} = (f \bmod (X^2 - \zeta^{2\text{BitRev7}(0)+1}), \dots, f \bmod (X^2 - \zeta^{2\text{BitRev7}(127)+1}))$$

Dans l'algorithme du dessus $\hat{f}$ est stocké dans un tableau de 256 entiers modulo q où :

$$f \bmod (X^2 - \zeta^{2\text{BitRev7}(l)+1}) = \hat{f}[2l] + \hat{f}[2l + 1]X$$

Avec i allant de 0 à 127.

La NTT ainsi réalisée, il est maintenant possible de multiplier les polynômes entre eux, ce qui correspond simplement à une multiplication termes à terme dans le domaine de la NTT.

C. Création d'un Schéma de chiffrement à clé publique (PKE)

i. Génération de clé

La génération de clés s'effectue avec l'algorithme K-PKE dont le fonctionnement est décrit ci-après.

La première étape consiste à générer de manière aléatoire 32 octets qui seront directement hachés par l'algorithme SHA3-512 et qui donnera en sortie 2 séries de 32 octets (ρ et σ) qui serviront de graine par la suite.

Nous créons ensuite une matrice $\mathbf{A}$ que nous remplissons à l'aide de l'algorithme SHAKE128 qui prend en paramètre la concaténation de ρ ainsi que des index i et j pour chaque coefficient de $\mathbf{A}$. On transforme chaque coefficient de la matrice avec la NTT. On obtient finalement une matrice $\hat{\mathbf{A}}$ appartenant aux entiers modulo $q = 3329$ de longueur 256 bits.

Nous créons ensuite 2 vecteurs de longueurs $\mathbf{k}$, appartenant également aux entiers modulo $q = 3329$ de longueur 256 bits, dont les coefficients sont définis par la distribution centrée binomiale, qui prend en paramètre un nombre η ainsi qu'un flux d'octets uniformément aléatoire. Ce flux d'octets est créé grâce à l'algorithme SHAKE256 qui prend en paramètres η , σ et $\mathbf{N}$. $\mathbf{N}$ est un nombre entier qui est incrémenté pour chaque coefficient à travers les 2 différents vecteurs, afin de ne pas en produire 2 identiques. Ces 2 vecteurs sont nommés $\mathbf{s}$ et $\mathbf{e}$.

Nous transformons par la suite $\mathbf{s}$ et $\mathbf{e}$ avec la NTT pour obtenir respectivement $\hat{\mathbf{s}}$ et $\hat{\mathbf{e}}$. Nous pouvons créer $\hat{\mathbf{t}} = \hat{\mathbf{A}} \times \hat{\mathbf{s}} + \hat{\mathbf{e}}$.

Afin d'obtenir la clé publique, nous retransformons $\hat{\mathbf{t}}$ en octets et le concaténons avec ρ . La clé privée quant à elle contient $\hat{\mathbf{s}}$.

$\hat{\mathbf{t}}$ contient donc la clé de chiffrement, $\mathbf{N}$ fait office d'entropie dans la création de $\hat{\mathbf{e}}$, qui est considéré comme du bruit.

La complexité augmente lorsque la valeur de k augmente car la taille de la matrice A ainsi que les vecteurs s et e augmentent. Le paramètre η ne sert que pour définir la distribution centrée binomiale.

Tous les flux d'octets générés aléatoirement doivent être générés par un algorithme approuvé par le NIST et une sécurité d'au minimum 128, 192 et 256 bits pour les algorithmes ML-KEM-512, ML-KEM-728 et ML-KEM-1024 respectivement.

ii. Chiffrement

Le chiffrement se fait grâce à une seule fonction centrale : K-PKE.Encrypt. Elle utilise des fonctions comme sampleNTT, XOF, SamplePolyCBD et PRF, fonctions qui ne seront pas détaillées ici étant donné qu'elles apportent uniquement des valeurs pseudo-aléatoires.

Ensuite, nous ne prendrons pas en considération les fonctions ByteDecode, Compress, Decompress, ni les distinctions entre t et $\hat{t}$, de même que pour toutes les variables affectées par le NTT. Ces éléments ne sont pas nécessaires pour expliquer l'algorithme à un niveau plus élevé. Cependant, les fonctions Compress et Decompress seront expliquées à la fin de cette section.

La fonction K-PKE.Encrypt requiert les paramètres suivants : la clé de chiffrement, un message de 32 octets et un vecteur r de 32 octets.

Pour plus de clarté, voici une explication des variables :

- $\hat{t}$ contient la clé de chiffrement
- $\hat{A}$ est une matrice de taille k (2,3 ou 4), contenant des polynômes aléatoires
- $\hat{r}$ est un vecteur de valeurs aléatoires faisant office de bruit
- N ajoute simplement de l'entropie pour éviter la répétition de polynômes
- $e1$ et $e2$ font également office de bruit
- $\hat{r}$, $e1$ et $e2$ sont donc bruyant (avec r ayant plus d'influence que $e1$ et $e2$)
- s est une variable secrète (clé de déchiffrement)
- μ contient le message à cacher

Ensuite, deux équations, u et v , entrent en jeu. Ces équations constituent la base de l'algorithme et rendent difficile le déchiffrement de la sortie sans la connaissance de la clé privée :

$$u = a * r + e1$$

$$v = tr + e2 + \mu$$

Ces deux équations fournissent des équivalences très difficiles à résoudre, avec une complexité de n^2 . Sachant que $t = A * s$ (déterminé grâce à la génération des clés) et en négligeant le bruit pour simplifier les calculs, on peut en déduire l'équation suivante :

$$t * r = s * u$$

Les valeurs u et v sont ensuite concaténées pour être envoyées. Le bruit se situe au niveau des bits de poids faible, ce qui limite les erreurs et n'affecte pas le résultat, toujours arrondi à l'unité la plus proche.

Voici une explication générale pour comprendre l'impact des fonctions Compress et Decompress :

- Compress supprime les x bits de poids faibles
- Decompress ajoute x bits à la fin, tous égaux à 0

Ainsi, $\text{Compress}(\text{Decompress}(y)) = y$. De cette manière, le bruit de LWE est encodé sur les bits de poids faibles ajoutés avec $\text{Decompress}()$, puis supprimés avec $\text{Compress}()$."

iii. Déchiffrement

La fonction K-PKE.Decrypt permet un déchiffrement rapide du message pour ceux qui possèdent la clé secrète. Explorons les opérations nécessaires pour récupérer le message d'origine.

Cette fonction prend en entrée deux paramètres : la clé de déchiffrement $\mathbf{s}$ et le texte chiffré $\mathbf{c}$. En premier lieu, le texte chiffré $\mathbf{c}$ est divisé en deux parties correspondant à $\mathbf{u}$ et $\mathbf{v}$ (leurs tailles respectives sont connues, ce qui simplifie cette opération).

Ensuite, la clé de déchiffrement $\mathbf{s}$ est utilisée sur $\mathbf{u}$ ($\mathbf{s} * \mathbf{u}$). On obtient donc $\mathbf{t} * \mathbf{r}$ (se référer à la fonction K-PKE.Encrypt pour plus de détails).

Pour rappel :

$$\mathbf{v} = \mathbf{t}\mathbf{r} + \mathbf{e}_2 + \mu$$

Il suffit donc de soustraire $\mathbf{v}$ à ce résultat pour obtenir μ (plus le bruit $\mathbf{e}_2$, mais celui-ci est éliminé avec les fonctions Compress et Decompress). Ensuite, μ est transformé pour le rendre lisible."

D. Conversion en Mécanisme d'encapsulation de clé (KEM)

i. Transformation FO

La transformation de Fujisaki-Okamoto (FO) constitue une protection majeure en cryptographie grâce à son renforcement contre les attaques à texte chiffré choisi (CCA) et les attaques à texte clair choisi (CPA). Ces attaques, extrêmement sophistiquées dans le domaine du chiffrement à clé publique, permettent à un adversaire de sélectionner des textes chiffrés et d'en obtenir les versions déchiffrées, ou inversement, risquant ainsi de compromettre la clé secrète ou la sécurité des données.

ii. Génération de clé

Dans le cadre du processus ML-KEM, le principe fondamental est le suivant : nous supposons que Bob et Alice souhaitent communiquer. Bob va commencer par générer une paire de clés, une pour le chiffrement et une pour le déchiffrement, à l'aide de l'algorithme K-PKE.KeyGen . Ensuite, il combine la clé de déchiffrement avec la clé de chiffrement, le hachage de cette dernière, et une valeur aléatoire. Tout cela est fait dans le but de faciliter la récupération de ces informations ultérieurement.

iii. Encapsulation

Alice utilise donc la clé de chiffrement afin de générer une clé partagée, avec deux fonctions de hachage qui retourneront la clé secrète partagé **K** et une valeur aléatoire **r** puis elle utilise l'algorithme K-PKE.Encrypt pour chiffrer son message et obtenir un ciphertext qu'elle va envoyer à Bob.

iv. Décapsulation

Bob récupère les différentes valeurs qu'il a stockées dans la clé de déchiffrement au préalable lors de la génération (la clé de déchiffrement, la clé de chiffrement, le hash de clé de chiffrement puis une clé de rejet). Avec la clé de déchiffrement, il utilise le K-PKE decrypt avec comme paramètres le ciphertext d'Alice et la clé de déchiffrement, pour obtenir **m'** qui est le ciphertext déchiffré. Ensuite pour se protéger des CCA ou CPA Bob va chiffrer à nouveau **m'** en simulant l'encapsulation. Il va donc utiliser une fonction de hachage pour récupérer une clé partagée **K'** et une valeur aléatoire **r'**. Avec la valeur de rejet et le ciphertext, il génère une clé alternative. Enfin il chiffre de nouveau **m'** avec la clé de chiffrement et **r'** pour obtenir **c'**.

Si l'encapsulation a été réalisée correctement du côté d'Alice, Bob devrait obtenir deux valeurs, **c** et **c'**, qui sont identiques. Dans ce cas, il peut renvoyer la clé secrète partagée. Si ces deux valeurs sont différentes, cela indique un risque de modification du texte chiffré pendant la communication entre Alice et Bob. Dans ce cas, Bob renvoie la clé alternative au lieu de signaler une erreur explicite, afin de préserver la sécurité.

III. ML-DSA

Un autre protocole se voulant aussi résistant aux attaques des ordinateurs quantiques est encore en phase d'étude. Il s'appelle ML-DSA et est assez proche de ML-KEM, mais se distingue de ce dernier en étant un protocole de signature numérique. ML-DSA utilise un algorithme de machine learning afin de générer des clés publiques et privées. Le protocole fonctionne en suivant ces grandes étapes :

- L'expéditeur génère une clé publique et une clé privée à l'aide d'un algorithme de machine learning se nommant ML-DSA.KeyGen.
- L'expéditeur utilise la clé privée pour signer le document grâce à un algorithme se nommant ML-DSA.Sign
- Le destinataire utilise la clé publique pour vérifier la signature avec cette fois-ci un algorithme se nommant ML-DSA.Verify

ML-DSA utilise de nombreuses techniques similaires à ML-KEM comme la multiplication de polynômes en utilisant la NTT afin de simplifier les calculs. Nous pouvons observer que l'utilisation de la NTT est cette fois-ci plus facile car une racine 512^{ème} existe avec la valeur de q de ce protocole ($q = 8380417$).

La sécurité des schémas de signature numérique est liée à deux problèmes centraux : le problème d'apprentissage avec erreurs (LWE) comme vu dans le protocole ML-KEM et le problème de solution d'entier court (SIS). Ce problème consiste à trouver une solution à un système linéaire d'entiers donné, dont l'une des solutions est de petite taille. C'est un problème très efficace car il est difficile de trouver une solution au problème SIS, même si l'on possède un ordinateur très puissant. Cette difficulté est due au fait que le problème SIS est un problème d'optimisation non-linéaire.

Le protocole ML-DSA utilise des matrices à coefficients courts et un protocole de signature Fiat-Shamir avec abandons (permet au signataire d'abandonner le processus de signature s'il rencontre un problème). Les matrices sont structurées par module pour réduire la taille des clés et des signatures, et la matrice A est dérivée de manière pseudo-aléatoire à partir d'une graine publique pour réduire encore la taille de la clé publique afin de gagner en sécurité, rapidité et faciliter la distribution et le stockage de cette clé.

Tout comme ML-KEM, ce protocole est encore en phase de test et il est encore sujet à des modifications.

IV. Projection

Après avoir exposé le protocole et supposé sa compréhension, nous envisageons maintenant son implémentation. Pour maintenir notre approche didactique, privilégier un développement pédagogique semble la voie appropriée. L'objectif est de visualiser le fonctionnement des protocoles en ajustant les paramètres, permettant ainsi une compréhension détaillée des étapes tout au long du processus. Nous allons également introduire une représentation visuelle pour illustrer l'impact et l'intérêt de la NTT.

Dans cette démarche, certaines mesures de sécurité pourront être temporairement omises afin de faciliter la mise en place et la visualisation d'éventuelles vulnérabilités. Par exemple, la transformation FO pourra être exclue pour mettre en évidence son intérêt.

Les prochaines étapes consisteront à prendre des décisions d'implémentation, à choisir le langage le plus adapté, ainsi qu'à définir les bibliothèques et les algorithmes annexes nécessaires. Ces choix guideront la concrétisation de notre implémentation didactique du protocole.

Bibliographie

NIST, “*FIPS 203 (Draft)*”, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.ipd.pdf>, 20/09/23

NIST, “*FIPS 204 (Draft)*”, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.204.ipd.pdf>, 20/09/23

Langlois, A., Stehlé, D., “Worst-case to average-case reductions for module lattices”. *Des. Codes Cryptogr.* **75**, 565–599 (2015). <https://doi.org/10.1007/s10623-014-9938-4>, 03/12/23

Daniel O'Connor, “*The Quantum Frontier: NTT's Quest for Unbreakable Cryptography*”, https://group.ntt/en/magazine/blog/quantum_frontier/, 28/11/23

Sin-Wei Chiu, “NTT-Based Polynomial Modular Multiplication for Homomorphic Encryption: A Tutorial”, <https://arxiv.org/pdf/2306.12519.pdf>, 21/06/23

Groupe Projet, “Suivi projet S1” - Trello, <https://trello.com/invite/b/zTCfR24c/ATTI8db847062e9928541d7bf472ffe7aa121F3F5C90/bibliographie>, 22/12/23