Université de Limoges

# Chiffrement Crystals Kyber

Travail effectué Par

Mohamed Lamine NDIONGUE

Sous la supervision du Professeur François Arnault

# Table des matières


# Introduction

Définie comme étant l'art de cacher de l'information dans des messages de par ses peformances et ses services d'usages sûrs, la cryptographie est une technique d'écriture où un message chiffré est écrit avec l'utilisation des codes secrets ou des clés de chiffrement. Sa mission est d'assurer la sécurité de par la disponibilité, la confidentialité, l'intégrité et la non-répudiation qui constituent les principes fondamentaux de son emploi. Ainsi, dès lors, il devient important pour les cryptosystèmes de respecter les normes établies et de garantir la sécurité et le bien-être des données. Pour ça, il serait important pour la cryptographie d'émerger et qu'elle se révolutionne selon des moments et des périodes auxquels elle est confrontée. De ce fait, depuis l'antiquité jusqu'à notre ère, elle ne fait que connaître et se faire connaître du nouveau dans son utilisation. Arrivée à la fin XIX$^{e}$ siècle, elle subie d'énorme menace notamment la révélation de l'algorithme de Peter Shor en 1994 sur la cryptographie à clé publique qui permettrait la factorisation des grands nombres en un temps polynômial et celle de l'algorihme de Grover en 1996 sur la cryptographie à clé symétrique qui donne une complexité $\sqrt{n}$ réduisant le temps de recherche d'une clé secrète. C'est dans ce contexte particulier marquant l'importante avancée de la technologie numérique plsu particulièrement du quantique que le NIST (National Institute Standard and Technology) a eu à lancer un appel à un concours en 2016 pour sélectionner les meilleurs cryptosystèmes capablent de résister contre les calculateurs quantiques. À l'issue de ce concours, en 2021, deux cryptosystèmes ont été standardisés qui sont l'algorithme de chiffrement Crystals-Kyber et l'algorithme de signature Crystals-Dilithium.

L'algorithme de chiffrement Crystals-Kyber, basé sur le MLWE (Modular Learning With Error), est un algorithme de Mécanisme d'Encapsulation de Clé (KEM) qui est passé de la sécurité IND-CPA (Indistinguability under Chosen Plaintext Attack) à la sécurité IND-CCA2 (Indistinguability under Chosen Chiphertext Attack) par la transformation de FUSAJAKI-OKAMOTO. Le problème de casser le chiffrement Crystals-Kyber repose sur la difficulté de résoudre le cryptosystème du problème d'apprentissage avec erreurs (LWE) sur les réseaux euclidiens.

## 0.1 Outils Mathématiques et préliminaires

### 0.1.1 Définition de MLWE (Modular Learning With Error)

Introduit en 2005 par Regev, LWE (Learning With Errors) est un problème dalgorithmique de la cryptographie basé sur les réseaux euclidiens et dont la sécurité repose sur la résistance contre un attaquant qui disposerait d'une machine quantique. Ainsi, son utilité devient prometteux dans le but de construire des primitives cryptographiques notamment le chiffrement. Lors de sa procédure d'utilisation, il question de retrouver un vecteur $\mathbf{s} \in \mathbb{Z}_q^n$ à partir d'un produit scalaires bruités de nombre aléatoire $s$ et des vecteurs connus $a_i \in \mathbb{Z}_q^n$ choisis uniformément et $n, q$ et $\alpha \in [0, 1]$ qui représentent des paramètres du problème [FaLa]. Il constitue un constitue un problème qui dispose différentes variantes parmi lesquelles nous avons : le problème Module d'apprentissage par erreurs (MLWE : Module Learning With Error).

### 0.1.2 Définition de la fonction NTT (Number Theoretic Transform)

Soit un nombre premier $q = 3329$ avec $q - 1 = 3328 = 2^8 \times 13$, $\mathbb{Z}_q$ le corps de base contenant 252-ième racine primitive de l'unité. Ce qui signifie, par opposition 512-ième racine primitve n'est pas inclue. Donc, le polynôme $X^{256} + 1$ se factorise en 128 polynômes de degré 2. Ainsi, soit $\varsigma = 17$, la première 256-ième racine primitive de l'unité modulo $q$ et la famille $\{\varsigma, \varsigma^3, \ldots, \varsigma^{256}\}$ qui constitue l'ensemble de toutes 256-ième racines de l'unité. Le polynôme $X^{256} + 1$ s'écrit comme [KyAlgoSupDoc] :

**Remarque importante sur la factorisation et les racines de l'unité**

Dans le cas de Kyber, le module est choisi tel que $q = 3329$. Ce choix implique que $q - 1 = 3328 = 2^8 \times 13$, et donc que $\mathbb{Z}_q$ contient une **racine primitive** 256**-ième de l'unité**, mais **pas de racine** 512**-ième de l'unité**.
Autrement dit, il existe un élément $\varsigma \in \mathbb{Z}_q$ tel que $\varsigma^{256} \equiv 1 \pmod q$, mais aucun élément ne satisfait $\omega^{512} \equiv 1 \pmod q$ avec $\omega^{256} \neq \pm 1$. C'est une propriété cruciale pour définir la **NTT (Number Theoretic Transform)** sur $R_q = \mathbb{Z}_q[X]/(X^{256} + 1)$. De cette structure découle la factorisation suivante :

$$X^{256} + 1 = \prod_{i=0}^{127} (X^2 - \varsigma^{2i+1}).$$

Ainsi, le polynôme $X^{256} + 1$ se factorise en **128 polynômes quadratiques**, et non en polynômes de degré $n = 256$. Chaque facteur correspond à une paire de racines conjuguées modulo $q$, ce qui permet la décomposition efficace utilisée par la NTT.

$$X^{256} + 1 = \prod_{i=0}^{127} X^2 - \varsigma^{2i+1} = \prod_{i=0}^{127} X^2 - \varsigma^{2br_7(i)+1}$$

où $br_7(i)$ pour $i \in \{0, \ldots, 127\}$ est l'inversion des bits des 7 bits non signés de l'entier $i$. Alors, la $NTT$ de $f \in R_q$ est donnée par :

$$(fmodX^2 - \varsigma^{2br_7(0)+1}, \ldots, fmodX^2 - \varsigma^{2br_7(127)+1}),$$

un vecteur de polynôme qui est par la suite sérialisé [1] en un vecteur dans $\mathbb{Z}_q^{256}$ d'une manière canonique.
Ainsi, la $NTT$ est définie comme suit :
$NTT : R_q \to R_q$, une bijection qui mappe $f \in R_q$ au polynôme avec de vecteur de coefficients susmentionnés.
Donc,

$$NTT(f) = \hat{f} = \hat{f}_0 + \hat{f}_1 X + \cdots + \hat{f}_2 55 X^{255}$$
$$\text{avec, } \hat{f}_{2i} = \sum_{i=0}^{127} f_{2j} \varsigma^{(2br_7(i)+1)j}$$
$$\hat{f}_{2i} = \sum_{i=0}^{127} f_{2j+1} \varsigma^{(2br_7(i)+1)j}$$

La représentation algèbrique naturelle de $NTT(f) = \hat{f}$ est dû du fait des 128 polynômes de degrès 1 utilisant la définition $\hat{f}_i$ des deux équations précédentes. Autrement dit,

$$NTT(f) = \hat{f} = (\hat{f}_0 + \hat{f}_1 X, \hat{f}_2 + \hat{f}_3 X, \ldots, \hat{f}_{254} + \hat{f}_{255} X).$$

Le produit $fg$ de $f, g \in R_q$ peut être effectué d'une manière efficace comme suit :
$NTT^{-1}(NTT(f) \circ NTT(g)$ où $NTT(f) \circ NTT(g) = \hat{f} \circ \hat{g} = \hat{h}$ et $NTT^{-1}$ est l'inverse de $NTT$.

**N.B. :**

La multiplication en Rq basée sur la transformation théorique des nombres (NTT) présente de nombreux avantages : elle est extrêmement rapide, ne nécessite pas de mémoire supplémentaire (comme, par exemple, la multiplication de Karatsuba ou de Toom) et peut être effectuée dans un espace de code très réduit. Par conséquent, il est devenu courant de choisir des paramètres de cryptographie basés sur des treillis pour prendre en charge cet algorithme de multiplication très rapide. Certains systèmes vont plus loin et intègrent le NTT dans la définition du système. Un exemple marquant est NewHope, qui échantillonne la valeur publique a dans le domaine NTT et envoie également les clés publiques et les textes chiffrés dans le domaine NTT afin d'économiser 2 NTT [KyAlgoSupDoc].

Pour l'échantillonnage uniforme dans $R_q$, Kyber utilise une approche déterministe pour échantillonner les éléments dans $R_q$ statiquement proches d'une distribution uniformément aléatoire. Ainsi, une fonction $Parse : \mathcal{B}* \to R_q$ qui prend en entrée un flux d'octets $B = b_0, b_1, b_2, \ldots$ et évalue la représentation $NTT, \hat{a} = \hat{a}_0 + \hat{a}_1 X + \cdots + \hat{a}_{n-1} X^{n-1} \in R_q$ de $a \in R_q$, définie dans cet échantillonnage.
Derrière cette fonction, si le tableau d'octets d'entrée est statiquement proche d'un tableau uniformément aléatoire, alors le polynôme de sortie est perçu statiquement plus proche d'un

1. Sérialiser : consiste, dans le langage informatique, à modifier des données parallèles pour les mettre en série à l'intérieur d'un flux

élément uniformément aléatoire de $R_q$. Donc ce polynôme de sortie représente un polynôme uniformément aléatoire dans $R_q$ en raison de la bijection de $NTT$ et de son mappage des coefficients uniformément aléatoire en polynôme. Ainsi, est défini l'algorithme de la fonction $Parse$ :

1. Entrée : $B = b_0, b_1, b_2, \ldots$, qui constitue un flux d'octets
2. Sortie : fonction de représentation $NTT\hat{a} \in R_q$ de $a \in R_q$.
   $i := 0$
   $j := 0$
   pour $j < n$ faire
   $d_1 := b_i + 256(b_{i+1} mod 16)$
   $d_2 := \lfloor b_{i+1}/16 \rceil + 16b_{i+e}$
   si $d_1 < q$ alors
   $\hat{a}_j := d_1$
   $j := j + 1$
   fin si si $d_2 < q$ et $j < n$ alors
   $\hat{a}_j := d_2$
   $j := j + 1$
   fin si
   $i := i + 3$
   fin pour
   retourne $\hat{a}_0 + \hat{a}_1 X + \cdots + \hat{a}_{n-1} X^{n-1}$

**Parse($B$)**

Cette fonction Parse($B$) qui reçoit en entrée un flux d'octets $B = b0, b1, b2, \ldots$ est utilisé dans le but d'effectuer un échantillonnage et elle calcule la représentation NTT $\hat{a} = \hat{a}_0 + \hat{a}_1 X + \hat{a}_2 X^2 + \cdots + \hat{a}_{n-1} X^{n-1} \in R_q$ of $a \in R_q$.
Donc L'intuition derrière la fonction Parse est que si le tableau d'octets d'entrée est statistiquement proche d'un tableau d'octets uniformément aléatoire, alors le polynôme de sortie est statistiquement proche d'un élément uniformément aléatoire de Rq. Il représente un polynôme uniformément aléatoire dans Rq car NTT est bijectif et permet donc d'associer des polynômes à coefficients uniformément aléatoires à des polynômes dont les coefficients sont à nouveau uniformément aléatoires [KyAlgoSup-Doc].

```python
def parse_B(B, q, n):
    a = [0] * n
    i = 0
    j = 0

    while j < n:
        d1 = B[i] + 256 * (B[i+1] % 16)
        d2 = (B[i+1] // 16) + 16 * B[i+2]

        if d1 < q:
            a[j] = d1
            j += 1

        if d2 < q and j < n:
            a[j] = d2
            j += 1

        i += 3

    return a
```

Ce code prend en entrée un flux d'octets $B$, une valeur $q$ et la taille $n$ de la représentation NTT, puis retourne la représentation NTT correspondante.

**$CBD_\eta(B = (b_0, b_1, \ldots, b_{64\eta-1}))$**

La définition pour l'échantillonnage d'un polynôme $f \in R_q$ selon $B_\eta$ de manière déterministe à partir de $64\eta$ octets de sortie et retourne une fonction aléatoire pour la spécification de Kyber qui permettra ainsi de définir le schéma de **Kyber.CPAPKE**. D'où la fonction $CBD$ (Contered Binomial Distribution) dont l'algorithme est défini comme suit :

$CBD_\eta : \mathcal{B}^{64\eta} \to R_q$

1. Entrée : Tableau d'octets $B = (b_0, b_1, \ldots, b_{64\eta})$
2. Sortie : Polynôme $f \in R_q$.
   $(\beta_0, \ldots, \beta_{512\eta-1}) := BytesToBits(B)$
   pour $i = 0$ à 255 faire
   $$a := \sum_{j=0}^{n-1} \beta_{2i\eta+j}$$
   $$b := \sum_{j=0}^{n-1} \beta_{2i\eta+n+j}$$
   $$f_i := a - b$$
   fin pour.
   retourne $f_0 + f_1 X + \cdots + f_{255} X^{255}$

```python
def BytesToBits(B):
    bits = []
    for byte in B:
        bits.extend([(byte >> i) & 1 for i in range(8)])
    return bits

def CBD_η(B, q, n, η):
    bits = BytesToBits(B)
    f = [0] * 256

    for i in range(256):
        a = sum(bits[2 * i * η : 2 * i * η + n])
        b = sum(bits[2 * i * η + n : 2 * i * η + 2 * n])
        f[i] = (a - b) % q

    return f
```

Cette implémentation prend un tableau d'octets $B$, une valeur $q$, une taille de polynôme $n$, et une valeur , puis retourne le polynôme $f$ selon les règles spécifiées dans votre algorithme.

**Fonctionnement de la fonction BytesToBits(B) :**
La fonction BytesToBits(B) prend un tableau d'octets B en entrée et retourne une liste de bits correspondant à la représentation binaire des octets dans le tableau.
Elle fonctionne comme suit :

1. Elle parcourt chaque octet dans le tableau d'octets B.
2. Pour chaque octet, elle parcourt les 8 bits de poids faible aux bits de poids fort.
3. Pour chaque bit, elle détermine s'il est défini à 0 ou à 1.
4. Elle ajoute ces bits à une liste qui contiendra finalement tous les bits de tous les octets du tableau d'octets.

Entre autre, Cette fonction est utilisée pour convertir un flux d'octets en une séquence de bits pour être utilisée dans le reste de l'algorithme.

**Définition de fonctions d'encodage et de décodage**

Kyber a besoin de sérialiser deux types de données en tableaux d'octets : les tableaux d'octets et les (vecteurs de) polynômes. Les tableaux d'octets sont trivialement sérialisés via l'identité, nous devons donc définir comment sérialiser et désérialiser les polynômes. Dans l'algorithme suivant.

Nous avons suivant : l'algorithme de décodage qui désérialise un tableau de 32 octets en un polynôme $f = f_0 + f_1X + \cdots + f_{255}X^{255}$.

$Decode_l(B = (b_0, b_1, b_2, \ldots, b_{256}))$ $Decode_l : \mathcal{B}^{32l} \rightarrow R_q$

1. Entrée : Tableau d'octets $B \in \mathcal{B}^{32l}$.
2. Sortie : Polynôme $f \in R_q$. $(\beta_0, \ldots, \beta_{256l-1}) := BytesToBits(B)$.
   pour $i = 0$ à 255 faire
   $$f_i := \sum_{j=0}^{l-1} \beta_{il+j^{2j}}$$
   fin pour
   retourne $f = f_0 + f_1 X + \cdots + f_{255} X^{255}$

La fonction d'encodage $Encode_l$ est définie comme étant l'inverse de $Decode_l$. Donc à chaque fois qu'il y a un encodage d'un vecteur de polynômes, alors il y a encodage individuel de chaque polynôme et la sortie constitue les tableaux d'octets.

Cette fonction prend un tableau d'octets $B$, une valeur $q$, et une valeur $l$, puis retourne le polynôme $f$ selon les règles spécifiées dans votre algorithme. La fonction BytesToBits est utilisée pour convertir le tableau d'octets en une séquence de bits. Ensuite, la fonction parcourt les bits pour calculer les coefficients du polynôme $f$.

```python
def BytesToBits(B):
    bits = []
    for byte in B:
        bits.extend([(byte >> i) & 1 for i in range(8)])
    return bits

def Decodel(B, q, l):
    bits = BytesToBits(B)
    f = [0] * 256

    for i in range(256):
        fi = sum(bits[i * l : i * l + l])
        f[i] = fi % q

    return f
```

### 0.1.3 Transformation de Fujasaki-Okamoto

La transformation Fusajaki-Okamoto est une méthode d'utilisation permettant la construction de cryptosystèmes passant de la sécurité du schéma PKE IND-CPA à la sécurité du schéma KEM IND-CCA (CCA2) dont la finalité serait d'appliquer l'échange de clé KEX consistant à établir une clé secrète commune pour assuer la bonne sécurité de communication.

**KEX(Échange de clé = Key Exchange)**

KEX est un algorithme d'échange de clé décrivant ainsi le processus d'exécution d'établissement d'une clé secrète commune entre deux entités.

**PKE(Chiffrement à Clé Publique = Publique Key Encryption en anglais)**

Un schéma de chiffrement à clé publique est un cryptosystème composé de trois algorithmes :

- Un algorithme de génération de clé retournant ainsi deux clés : une clé publique $p_k$ et une clé secrète $s_k$ ;
- Un algorithme de chiffrement d'un message donné avec usage de la clé publique $p_k$ qui permet de calculer le chiffré ;
- Un algorithme de déchiffrement qui retourne le message clair initial en calculant le chiffré avec la clé secrète $s_k$.

**KEM (Mécanisme d'Encapsulation de Clé = Key Encryption Mechanism en anglais)**

Il s'agit d'un cryptosystème, tout comme le PKE, qui est composé de tois algorithmes tels que :

- Algorithme de génération de clé : qui génère deux clés : clé publique $p_k$ et clé secrète $s_k$ ;
- Algorithme d'Encapsulation : qui à partir de l'usage de la clé publique, calculer une clé de session [a] et le texte chiffré ;
- Algorithme de décapsulation de clé : prenant en entrée un texte chiffré et une clé privée pour ensuite calculer une clé de session dont l'égalité aurait une forte probabilité dépendant ainsi de l'authenticité de la clé secrète utilisé.

*a.* Une clé de session est toute clé cryptographique symétrique utilisée pour chiffrer une seule session de communication.

Toutes ces primitives sont liées, car la construction des KEM peut être effectuée à partir des PKE et celle des KEX à partir des KEM et aussi à partir des PKE. Après un long processus de discussion, le NIST a décidé de normaliser les KEM. Le choix s'est porté sur KEX, car tous les candidats KEX disponibles nécessitaient une interaction et étaient donc moins généraux que PKE et KEM. Les KEM ont été préférés aux PKE car, dans la plupart des applications, l'objectif principal des PKE est de chiffrer les clés de session. Un KEM fait essentiellement la même chose, à la différence que le système contrôle le choix de la clé de session. Cette petite différence facilite l'élaboration de systèmes sûrs [WaBe].

### Notions de sécurité IND-CPA et IND-CCA

La définition des notions de sécurité IND-CPA et IND-CCA sont indispensable pour l'établissement de la transformation Fujasaki-Okamoto.

**IND-CPA (INDistinguishability under Chosen Plaintext Attack)**

Impliquant un chiffrement aléatoire, l'indiscernabilité par attaque au clair choisi est une sécurité passive décrivant une configuration dans laquelle un adversaire, connaissant la clé publique, peut lui-même chiffrer des messages. Cette notion de sécurité fait l'objet de l'indivulgation de toute information sur le texte en clair à l'adversaire. Tout doit être caché pour lui et seul le chiffré lui doit être parvenu. Donc ce qui donne à l'adversaire de faire le choix de deux messages dont l'un d'eux est chiffré et lui est remis. Cette sécurité IND-CPA n'est utilisable qu'avec un PKE probabiliste où l'algorithme de chiffrement est aléatoire [WaBe].

Elle est définie comme un jeu entre un adversaire et un challenger dans le cas du chiffrement à clé publique. [MaVe]

1. Tout d'abord, le challenger génère une paire de clés de chiffrement et envoie la clé publique $p_k$ à l'adversaire et garde la clé secrète $s_k$ ;
2. Ensuite, l'adversaire sélectionne une paire de messages $m_0$, $m_1$(de longueur égale) et les envoie au challenger ;
3. Le challenger choisit un bit aléatoire $b \in \{0, 1\}$ et crypte l'un des deux messages sous la forme $C^* \leftarrow \mathsf{Encrypt}(pk, M_b)$. Cela renvoie $C^*$ à l'adversaire ;
4. Finalement, l'adversaire émet une supposition $b'$. Nous disons que l'adversaire « gagne » s'il devine correctement : c'est-à-dire si $b' = b$.

**OW-CPA (One-waynes under Chosen Plaintext Attack)**

OW-CPA est un système qui décrit une situation dans laquelle un adversaire peut lui-même chiffrer des messages. Mais, étant donné que le message original n'a pas été choisi par l'adversaire, donc la sécurite indique qu'il lui serait impossible de retrouver le message initialement transmis. Cette notion de sécurité semble plus faible par rapport à celle de IND-CPA mais sa réalisation peut être effectuée grâce au PKE déterministe (dPKE) où le chiffrement se fait sous clé publique fixe donnant ainsi le même texte chiffré [WaBe].

**IND-CCA (INDistinguishability under Chosen Ciphertext Attack)**

L'Indiscernabilité par attaque de textes chiffrés est un système actif décrivant la configuration dans laquelle un adversaire en plus d'être capable de génèrer des encapsulations, peut aussi disposer des moyens pour apprendre la décapsulation des textes chiffrés. Sa sécurité est décrite comme, étant donné un chiffré, toute information sur information sur la clé encapsulée, à l'exception de la clé de session de longueur différente, doit être cachée à l'adversaire. Donc après modélisation, l'adversaire génère un texte chiffré et une clé de session soit renvoyée par l'encapsulation après génèration du texte chiffré, soit échantillonnée de façon aléatoire et indépendante du texte chiffré. Et qu'au final, c'est à l'adversaire de déterminer laquelle des deux est la bonne [WaBe].

### Tranformation Fusajaki-Okamoto

- **Étape 1 (IND-CPA PKE vers OW-CPA dPKE)** : Dans cette partie pour la transformation F.O, l'idée est rendre le système le système PKE moins aléatoire, c'est-à-dire définir un algorithme aléatoire en un simple algorithme déterministe, prenant en plus un caractère aléatoire sous la forme d'une chaîne de bits en entrée. Donc la rendomisation s'effectue en remplaçant l'entrée aléatoire par le hachage du message.
  Cette transformation permet de convertir un schéma IND-CPA PKE en un schéma OW-CPA dPKE en combinant le chiffrement à clé publique avec une fonction de hachage cryptographiquement sûre. [WaBe]

- **Étape 2 (OW-CPA dPKE vers IND-CCA KEM)** : l'implication OW-CPA dPKE vers IND-CCA KEM indique qu'un système qui satisfait aux exigences de sécurité OW-CPA et utilise un chiffrement dPKE peut également garantir la sécurité IND-CCA, et peut même inclure un mécanisme KEM pour un échange de clé sécurisé. Elle montre qu'étant donné un texte chiffré, le résultat obtenu d'une opération de chiffrement, est un chiffrément testant ainsi s'il est possible d'obtenir le même texte chiffré. Ce principe de fonctionnement du schéma PKE permet d'ajouter du contrôle de rechiffrement dans la routine de déchiffrement. La conséquence de cet fonctionnement est tout texte chiffré n'étant pas choisi en chiffrant un message (et sans connaissance de la clé privée) sera rejeté, car il échoue à la vérification du rechiffrement.
  Le schéma dPKE est ensuite transformé en un KEM comme suit. La génération des clés reste la même. L'algorithme d'encapsulation choisit un message aléatoire à chiffrer avec le schéma dPKE. Le texte chiffré sous le schéma dPKE est le texte chiffré KEM. La clé de session est le hachage de la concaténation du message et du texte chiffré. L'algorithme de décapsulation déchiffre le texte chiffré et exécute le contrôle de rechiffrement. Si la vérification échoue, l'algorithme renvoie une clé de session aléatoire. Si la vérification réussit, l'algorithme renvoie le hachage du message déchiffré et du texte chiffré en tant que clé de session [WaBe].

**Remarque**

Un mécanisme d'encapsulation de clé (KEM) est utilisé pour envoyer une clé symétrique entre deux parties à l'aide d'algorithmes asymétriques. Dans cette méthode, l'expéditeur encapsule la clé symétrique dans un texte chiffré à l'aide de la clé publique du destinataire. À la réception du texte chiffré, le destinataire décapsule et récupère la clé symétrique à l'aide de sa clé privée, garantissant ainsi un échange sécurisé et authentifié sans partager directement la clé symétrique pendant la transmission [UdPa].

## 0.2 Algoritmhes de Kyber

CRYSTALS-Kyber est basé sur le problème d'apprentissage avec erreurs (LWE) sur les réseaux modulaires. L'algorithme implique les étapes suivantes :

- Encapsulation de la clé publique :

1. Alice génère une clé publique et une clé privée en utilisant des paramètres de réseau modulaire spécifiques.
2. Alice envoie la clé publique à Bob.

- Encapsulation de la clé partagée :
  1. Bob génère un message aléatoire et l'encapsule à l'aide de la clé publique d'Alice.
  2. Bob envoie le message encapsulé et des informations supplémentaires à Alice.
- Décapsulation de la clé partagée :
  1. Alice utilise sa clé privée et les informations supplémentaires de Bob pour déchiffrer le message encapsulé et récupérer la clé partagée.
  2. Alice vérifie l'authenticité du message encapsulé pour s'assurer qu'il provient de Bob.

  Les étapes ci-dessus permettent à Alice et Bob d'établir une clé partagée secrète qu'ils peuvent utiliser pour chiffrer des communications ultérieures.

**N.B :** La conception de Kyber fait intervenir l'usage de fonctions spécifiques telles que :

- une fonction pseudo aléatoire $PRF$ (Pseudo Random Function en anglais) définie par :

$$PRF : \mathcal{B}^{32} \times \mathcal{B} \to \mathcal{B}^*$$

  La fonction PRF Kyber est basée sur le problème de la complexité de l'apprentissage avec erreurs (MLWE). Cette fonction PRF fonctionne comme suit :
  - ◇ Les deux parties, l'émetteur et le destinataire, génèrent un ensemble de paramètres MLWE ;
  - ◇ L'émetteur génère un vecteur aléatoire, appelé le vecteur d'encapsulation ;
  - ◇ L'émetteur applique la fonction PRF au vecteur d'encapsulation. La fonction PRF renvoie un vecteur chiffré, appelé la clé de session.

```python
import hashlib

def PRF_Kyber(ke, pk):
    # Vérifier le type de ke
    if isinstance(ke, str):
        # Si ke est une chaîne de caractères, l'encoder en bytes
        ke_bytes = ke.encode()
    elif not isinstance(ke, bytes):
        # Si ke n'est ni une chaîne de caractères ni un objet bytes, lever une erreur
        raise TypeError("La clé secrète doit être une chaîne de caractères ou un objet bytes")
    else:
        # Si ke est déjà un objet bytes, pas besoin de l'encoder
        ke_bytes = ke

    # Concaténation de la clé secrète (ke) et du vecteur d'encapsulation (pk)
    data = ke_bytes + pk

    # Utilisation de la fonction de hachage SHA-256 pour obtenir la clé de session
    hashed_data = hashlib.sha256(data).digest()

    return hashed_data
```

  Ici la fonction PRF_Kyber prend une clé secrète sous forme de chaîne de caractères ou d'objet bytes sans générer d'erreurs. Si la clé secrète est déjà en bytes, elle est simplement utilisée telle quelle, sinon elle est encodée en bytes avant d'être concaténée avec le vecteur d'encapsulation.

- une fonction de sorite extensible $XOF$ (eXtensible Output Function) définie par :

$$XOF : \mathcal{B}^{32} \times \mathcal{B} \times \mathcal{B} \to \mathcal{B}^*$$

La fonction XOF est une fonction simple qui combine deux vecteurs en un seul vecteur.

```python
def XOF(vecteur1, vecteur2, cle_secrete):
    # Vérification des longueurs des vecteurs
    if len(vecteur1) != len(vecteur2):
        raise ValueError("Les vecteurs doivent avoir la même longueur")

    # XOR des deux vecteurs
    resultat = [a ^ b for a, b in zip(vecteur1, vecteur2)]

    # XOR du résultat avec la clé secrète
    resultat = [a ^ b for a, b in zip(resultat, cle_secrete)]

    return bytes(resultat)  # Convertir la liste en bytes

# Exemple d'utilisation
vecteur1 = b'\x01\x02\x03\x04'  # Premier vecteur
vecteur2 = b'\x05\x06\x07\x08'  # Deuxième vecteur
cle_secrete = b'\x09\x0A\x0B\x0C'  # Clé secrète

resultat = XOF(vecteur1, vecteur2, cle_secrete)
print("Résultat de XOF:", resultat)

## on convertit resultat en décimal comme suit :
vecteur1_decimal = int.from_bytes(resultat, byteorder='little')

print("Vecteur en décimal:", vecteur1_decimal)
```

Ainsi, la fonction XOF prend deux vecteurs et une clé secrète en entrée, puis elle effectue un XOR entre les deux vecteurs et ensuite elle effectue un XOR entre le résultat et la clé secrète. La sortie est ensuite retournée sous forme de bytes.

- des fonctions de hachages $H$ et $G$ définies par :

  $$H : \mathcal{B}^* \to \mathcal{B}^{32} \text{ et } G : \mathcal{B}^* \to \mathbb{B}^{32} \times \mathcal{B}^{32}.$$

  la fonction H prend un message en entrée et renvoie son hachage SHA-256 (représenté sous forme de bytes).
  La fonction G prend également un message en entrée, calcule son hachage SHA-256, puis renvoie les 32 premiers octets et les 32 derniers octets de ce hachage digest, représentés comme deux objets de type bytes.

```python
import hashlib

def H(message):
    # Créer un objet de hachage SHA-256
    h = hashlib.sha256()

    # Mettre à jour le hachage avec le message
    h.update(message)

    # Renvoyer le hachage digest (32 octets)
    return h.digest()

def G(message):
    # Créer un objet de hachage SHA-256
    h = hashlib.sha256()

    # Mettre à jour le hachage avec le message
    h.update(message)

    # Renvoyer les 32 premiers octets du hachage digest
    h1 = h.digest()[:32]

    # Renvoyer les 32 derniers octets du hachage digest
    h2 = h.digest()[32:]

    return h1, h2
```

- une fonction de dérivation de clé $KDF$ (Key Derivation Function) définie par :

$$KDF : \mathcal{B}^* \to \mathcal{B}^*$$

La fonction KDF Kyber est basée sur la fonction PRF. Elle est utilisée pour dériver une clé de chiffrement à partir de la clé publique et de la clé privée. Elle fonctionne comme suit :

  - ◇ Les deux parties, l'émetteur et le destinataire, génèrent un ensemble de paramètres MLWE ;
  - ◇ L'émetteur applique la fonction KDF à la clé publique et à la clé privée. La fonction KDF renvoie un vecteur chiffré, appelé la clé de chiffrement ;

```python
import PRF_Kyber as PRF
def KDF_Kyber(cle_publique, cle_privee):
    # Utilisation de la fonction PRF pour dériver la clé de chiffrement
    cle_de_chiffrement = PRF.PRF_Kyber(cle_publique, cle_privee)
    return cle_de_chiffrement

# Exemple d'utilisation
cle_publique = b'cle_publique'
cle_privee = b'cle_privee'

cle_de_chiffrement = KDF_Kyber(cle_publique, cle_privee)
print("Clé de chiffrement dérivée:", cle_de_chiffrement)
```

Cette fonction KDF_Kyber utilise la fonction PRF_Kyber de la bibliothèque PRF_Kyber (importée sous le nom PRF) pour dériver une clé de chiffrement à partir d'une clé publique et d'une clé privée dans le contexte du schéma de chiffrement Kyber.

- La fonction CBD Kyber est utilisée pour chiffrer les données en blocs. Elle est basée sur la fonction XOR. Cette fonction CBD fonctionne comme suit :
  - ◇ Le bloc de données à chiffrer est divisé en plusieurs blocs plus petits ;
  - ◇ Chaque bloc est chiffré en utilisant la clé de chiffrement ;
  - ◇ Les blocs chiffrés sont ensuite recombinés pour former le bloc de données chiffré.

où $\mathcal{B}$ est défini comme l'ensemble $\{0, \ldots, 255\}$, des 8 bits d'entiers non signés, $\mathcal{B}^k$ représente l'ensemble des tableaux d'octets de longueur $k$ et $\mathcal{B}^*$ représente l'ensemble des tableaux d'octets de longueurs arbitraires (ou de flux).

Et nous voyons aussi qu'avec le cryptosytème Crystals-Kyber, il est possible d'effectuer un protocole d'échange de clés, donc basé sur la dureté du Module-LWE dans les modèles de l'oracle aléatoire classique et quantique comme l'ont souligné **Bos** et **al.** dans [CCAKEM] à la page 8. Dans cette rubrique, un jeu de protocole d'échange de clés et un jeu de sécurité ont été effectués.

### 0.2.1 Algorithmes Kyber.CPAPKE

Le schéma Kyber.CPAPKE est similaire au schéma de chiffrement LPR introduit par Lyuba-shevsky, Peikert, et Regev dans [LyPeOd] mais leur seule différence est que Kyber.CPAPKE utilise le Module-LWE au lieu de Ring-LWE utilisé par le système LPR.

**Parameters.** Ces paramètres $n, k, q_{,1}\ _{,2}, d_u$ et $d_v$ sont utilisés dans le schéma Kyber.CPAPKE tels que $n = 256$ et $q = 3329$. Nous avons l'algorithme de génération de clé suivant :

⊙ Génération de clés GenCPA_Key : Algorithme de génération de clé $Kyber.CPA.KeyGen()$ : qui génère une paire de clés construite à partir de la fonction d'encodage $Encode_l$ avec $l = 12$ de paramètres de concaténation $\hat{t}modq$ et d'un nombre $\rho$ aléatoirement tiré de $G(d)$ qui donne en sorite $pk$ et la clé secrète $sk$ est donnée en sortie par la fonction $Encode_l$ de paramètre $\hat{(}s)modq$. Ainsi, il est algorithmiquement défini comme suit :

**Kyber.CPAPKE.KeyGen()**

1. Sortie : La clé secrète $sk \in \mathcal{B}^{12kn/8}$.
2. Sortie : la clé publique $pk \in \mathcal{B}^{12kn/8+32}$
   $d \leftarrow \mathcal{B}^{32}$

   $(\rho, \sigma) := G(d)$
   $N := 0$
   #Génération de la matrice $\hat{A} \in R_q^{k \times k}$ dans le domaine $NTT$
   pour $i = 0$ à $k - 1$ faire
   pour $j = 0$ à $k - 1$ faire
   $\hat{A}[i][j] := Parse(XOF(p, j, i))$
   fin pour
   fin pour
   pour $i = 0$ à $k - 1$ faire
   # Echantillonner $s \in R_q^k$ à partir de $B_{\eta 1}$
   $s[i] := CBD_\eta(PRF(\eta, N))$
   $N := N + 1$
   fin pour

pour $i = 0$ à $k - 1$ faire
    # Echantillonner $e \in R_q^k$ à partir de $B_{\eta 1}$
    $e[i] := CBD_{\eta 1}(PRF(\sigma, N))$
    $N := N + 1$
fin pour
$\hat{s} := NTT(s)$
$\hat{e} := NTT(e)$
$\hat{t} := \hat{A} \circ \hat{s} + \hat{e}$
$pk := (Encode_{12}(\hat{t} \bmod q)||q)$
$sk := (Encode_{12}(\hat{s} \bmod q)$
retourne $(pk, sk)$, avec $pk := As + e$ et $sk = s$.

**Description de l'algorithme :** Cet algorithme de génération de clés secrètes $s_k$ et publique $p_k$ se décrit comme suit :

1. Initialisation :
   - $d$ est un vecteur de 32 bits tirés aléatoirement de $\mathcal{B}^{32}$ ;
   - $\rho$ et $\sigma$ sont obtenus par l'application de la fonction de hachage $G$ sur le vecteur d ;
   - $N$ est initialisé à 0.
2. Génération de la matrice $\hat{A}$, par $\hat{A}[i][j]$ pour $i$ de 0 à $k-1$ et pour $j$ de 0 à $k-1$ avec l'utilisation de la fonction $Parse$ appliquée sur une sortie de la fonction $XOR$ ;
3. Echantillonnage de $s$ par $s[i]$ pour $i$ de 0 à $k - 1$ à partir d'une distribution $B_{\mu 1}$ en utilisant une fonction $CBD_\mu$ appliquée sur une sorite de la fonction PRF de paramètres $\sigma$ et $N$ ;
4. Echantillonnage de $e$ : par $e[i]$, pour $i$ de 0 à $k - 1$ à partir d'une distribution $B_{\mu 1}$, par l'utilisation d'une fonction $CBD_\mu$ appliquée sur une sortie de la fonction PRF de paramètres $\sigma$ et $N$ ;
5. Transformation du vecteur $s$ et du vecteur d'erreur $e$ par la fonction de **T**ranformation **T**héorique des **N**ombes (Numbers TheoricalTransform). Et le caclul de $\hat{t}$ par $\hat{A}$ multipliée par $\hat{s}$ plus $\hat{e}$ ;
6. Encodage $pk$ et $sk$ : Encoder $\hat{t}$ et $\hat{s} mod^+ q$ par l'utilisation d'encodage 12.
   - $pk := (Encode_{12}(\hat{t} \bmod q)||q)$ ;
   — $sk := (Encode_{12}(\hat{s} \bmod q)$ ;
7. Retourne la paire de clés $(pk, sk)$, où $pk$ est la clé publique et $sk$ est la clé privée.

⊙ Chiffrement EncCPA : Algorithme de chiffrement CPA Kyber $Kyber.CPA.Enc(pk = (t, \rho), m \in \mathcal{M})$ : qui donne en sortie le chiffré $c = (c_1, c_2)$ tel que $c_1 = Encode_{d_u}(Compress_q(u, d_u))$ et $c_2 = c_1 = Encode_{dv}(Compress_q(v, d_v))$ où $\mathbf{r} \in R_q^k$ et $e_1 \in R_q^k, e_2 \in R_q$. Il est décrit comme suit :

**Kyber.CPAPKE.Enc**$(pk, m, r)$

1. Entrée : Clé publique $pk \in B^{12kn/8+32}$
2. Entrée : Message $m \in \mathcal{B}^{32}$
3. Entrée : Elément aléatoire $r$ (pour random coin) avec $r \in \mathbb{B}^{32}$

4. Sortie : Texte chiffré $c \in \mathbb{B}^{d_u kn/8+d_v n/8}$
   $N := 0$

   $\hat{t} := Decode_{12}(pk)$
   $\rho := pk + 12kn/8$
   pour $i = 0$ à $k-1$ faire
   Génération de la matrice $\hat{A} \in R_q^{k\times k}$ dans le domaine $NTT$
   pour $j = 0$ à $k-1$ faire
   $\hat{A}^T[i][j] := Parse(XOR(\rho, i, j))$
   fin pour
   fin pour
   pour $i = 0$ à $k-1$ faire
   Echantillonner $r \in R_q^k$ à partir de $r \in R_q^k$ à partir de $B_{\eta_1}$
   $r[i] := CBD_{\eta 1}(PRF(r, N))$
   fin pour
   pour $i = 0$ à $k-1$ faire
   #Echantillonner $e_1 \in R_q^k$ à partir de $B_{\eta_2}$
   $e_1[i] := CBD_{\eta_2}(PRF(r, N))$
   $N := N + 1$
   fin pour
   #Echantillonner $e_2 \in R_q$ à partir de $B_{\eta_2}$
   $e_2 := CBD_{\eta_2}(PRF(r, N))$
   $\hat{r} := NTT(r)$
   # $u := A^T r + e_1$
   $u := NTT^{-1}(\hat{A}^T \hat{r}) + e_1$
   # $v := t^T r + e_2 + Decompress_q(m, 1)$
   $v := NTT^{-1}(t^T \circ \hat{r}) + e_2 + Decompress_q(Decode_1(m), 1)$
   $c_1 := Encode_{du}(Compress_q(u, d_u))$
   $c_2 := Encode_{dv}(Compress_q(v, d_v))$
   retourne $c = (c_1||c_2)$ $\#c := (Compress_q(u, d_u), Compress_q(v, d_v)$

**Description du code :**

1. Initialisation :
   - initialisation du compteur $N$ à 0 ;
   - Décodage de la clé publque $pk$ pour retrouver $\hat{t}$ généré dans l'algorithme $KeyGen$ ;
   - Définition de $\rho$ comme $pk$ + une certaine valeur $\frac{12km}{8}$ ajoutée.
2. Génération de la matrice $\hat{A}$ : pour chaque paire $(i, j)$ dans la plage $k\times k$. Génération d'éléments de $\hat{A}$ dans le domaine $NTT$ avec l'utilisation de la fonction $XOR$ par la fonction $Parse$ qui renvoie la transposée.
3. Echantillonnage de $r$ dans $R_q^k$ à partir de $B_{\mu 1}$ à l'aide de la fonction $CBD_{\mu 1}$, qui prend en entrée une sortie de la fonction $PRF$.
4. Echantillonnage de $e_1$ dans $R_q^k$ à partir d'une distribution discrète $B_{\mu 2}$ à l'aide de la fonction $CBD_{\mu 2}$ prend ainsi en entrée une sortie de la fonction $PRF$.
5. Echantillonnage de $e_2$ dans $R_q$ à partir de $B_{\mu 2}$ à l'aide de la fonction $CBD_{\mu 2}$ qui prend en entrée la fonction $PRF$ et transformation du vecteur $r$ par la fonction $NTT$.
6. Calcul de $u$ et $v$ :

— calcul de $u$ en multipliant la transposée de $\hat{A}$ par $\hat{r}$ dans $R_q$ et en ajoutant $e_1$.
— calcul $v$ en multipliant $\hat{t}^T$ par $\hat{r}$ dans l'anneau $R_q$, en ajoutant $e_2$ et plus la décompression de la fonction décodage sur $m$.

7. Encodage des textes chiffrés :
   - Encodage $Encode_{d_u}(Compress_q(u, d_u))$ ;
   - Encodage $Encode_{d_v}(Compress_q(v, d_v))$.
8. Retourner la concaténation $c_1||c_2$ qui constitue le chiffré $c$.

**Remarque** :
Dans cet algorithme, la trappe est dûe à l'impossibilité pour un adversaire de récupérer le message original sans la clé secrète correspondante. C'est-à-dire, la difficulté pour un attaquant de trouver le message original vient des étapes suivantes décrites :

- **Détermination de** $\hat{A}$, étant donné une clé publique $pk$ fournie seule disponible, il est difficile pour un attaquant de déterminer la matrice $\hat{A}$ créée lors de l'algorithme de génération.
- **Récupération de** $r$ : Même disposant du chiffré $c$, il existe une probabilité très faible (négligeable) qu'un attaquant puisse déterminer $r$.
- **Calcul de** $u$ **et** $v$ : Dans ce contexte aussi même si un attaquant dispose $\hat{A}$ et $r$, il va être très difficile pour lui de calculer $u$ et $v$, car il y a d'ajouts de vecteurs d'erreurs $e_1$ et $e_2$ générés aléatoirement dans respectivement $R_q^k$ et $R_q$.

⊙ Déchiffrement DecCPA : Algorithme de déchiffrement CPA Kyber $Kyber.CPA.Dec(sk = s, c = (u, v))$ : est un algorithme dans lequel on décompresse le message encodé $c_1$ et $c_2$ pour ensuite retourner le nouveau message compressé ensuite encodé. Nous avons :

**Kyber.CPAPKE.Dec**$(sk = s, c = (c_1||c_2))$

1. Entrée : Clé secrète $sk \in \mathcal{B}^{12kn/8}$
2. Sortie : Texte chiffré $c \in \mathbb{B}^{d_v kn/8+d_v n/8}$
3. Sortie : Message $m \in \mathcal{B}^{32}$
   $u := Decompress_q(Decode_{du}(c), d_u)$

   $v := Decompress_q(Decode_{dv}(c + d_u kn/8), d_v)$
   $\hat{s} := Decode_{12}(sk)$
   $\#\ m := Compress_q(v - s^T u, 1)$
   $m := Encode_1(Compress_q(v - NTT^{-1}(\hat{s}^T \circ NTT(u)), 1))$
   retourn $m$.

**Description du code :** : Le processus de déchiffrement d'un texte chiffré $c$ pour retourner le message $m$ à partir de la clé secrète $s_k$ se fait comme suit :

1. **décompression de** $u$ **et** $v$ **:**
   - décompression de $u$ de la fonction décodage des paramètres $c$ et $d_u$ à base $q$.
   - décompression de $v$ de la fonction décodage des paramètres $c + d_u k\frac{n}{8}$ et $d_v$ à base $q$.
2. décompression de la clé secrète $s_k$ dans la base 12 pour obtenir la représentation $\hat{s}$ utilisée pour la suite de l'algorithhme.
3. récupération du message original $m$ à partir de l'encodage de la fonction compression $Compress_q(v - NTT^{-1}(\hat{s}^T \circ NTT(u)), 1)$.

### 0.2.2 Algorithmes CCAKEM

Kyber.CCAKEM est un schéma cryptographique permettant la construction d'un mécanisme d'encapsulation de clé KEM sécurisé IND-CCA à partir d'un PKE sécurisé IND-CPA par une transformation F.O. Le schéma consiste entre autre à effectuer :

1. **GenerationKeyCCAKEM** : un algorithme de génération de clé qui génère une paire de clés : publique $pk$ et secrète $sk$.

**Kyber.CCAKEM.KeyGen()**

(a) Sortie : clé publique $pk \in \mathbb{B}^{12kn/8+32}$
(b) Sortie : clé secrète $sk \in \mathbb{B}^{24kn/8/8+96}$
$z \leftarrow \mathcal{B}^{32}$
$(pk, sk) := Kyber.CPAPKE.KeyGen()$
retourne $(pk, sk)$.

**Description du code :** : Pour cet algorithme de génération de clé, on appelle la fonction génération de clé $Kyber.CPAPKE.KeyGen()$ pour avoir les clés publique $p_k$ et privée $s_k$ qui sont à leur tour retournées par cette fonction **Kyber.CCAKEM.KeyGen**.

2. **EncapsulationCCAKEM** : Un algorithme d'encapsulation : permettant d'encapsuler une clé secrète $K \in \mathcal{B}*$ à partir d'un message $m \in \mathcal{B}^{32}$ et d'une clé publique par des fonctions de hachage $H$ et $G$. Et donne en sortie le couple $(c, K)$ qui constitue l'encapsulé.

**Kyber.CCAKEM.Encaps($pk =$)**
# Les fonctions$PRF$, $XOF$, $H$, $G$ et $KDF$ sont définies précédemment.

(a) Entrée : Clé publique $pk \in \mathcal{B}^{12kn/8+32}$
(b) Sortie : Texte chiffré $c \in \mathcal{B}^{d_u kn/8+d_v n/8}$
(c) Sorite : Clé partagée $K \in \mathcal{B}^*$.
$m \leftarrow \mathcal{B}^{32}$

# Ne pas envoyer la sortie du système RNG (Random Number Generator).
$m \leftarrow H(m)$
$(\bar{K}, r) := G(m||H(pk))$
$c := Key.CPAKEM.Enc(pk, m, r)$
$K := KDF(\bar{K}||H(c))$
retourne $(c, K)$.

**Description du code :** : Cet algorithme décrit le processus d'encapsulation d'une clé partagée sous l'usage de la clé publique $p_k$. Ainsi, on les différentes étapes suivantes de l'algorithme :

(a) Génération d'un message aléatoire $m$ de 32 bits sur l'ensemble $\mathcal{B}^{32}$ assurant ainsi la diversité des messages à chaque exécution de l'algorithme.
(b) Hachage du message $m$ par l'opération $m \leftarrow H(m)$ pour obtenir un haché de $m$ utilisé ensuite dans les étapes suivantes. Cette fonction de hachage $H$ assure que $m$ a une forme standard et imprévisible.

(c) Génération de la clé partagée intermédiaire et du nonce [2] par l'utilisation de la fonction hachage $G$ ($(\bar{K}, r) := G(m||H(p_k))$ pour ensuite génèrer une clé $\bar{K}$ et un nonce $r$. Cette fonction $G$ prend en entrée la concaténation du message haché $m$ et du haché de la clé publique $p_k$ assurant ainsi que la clé partagée et le nonce sont dérivés de manière sécurisée à partir des entrées.

(d) L'encapsulation du message $m$ et le nonce $r$ par $c := Key.CPAKEM.Enc(p_k, m, r)$ est réalisée avec l'usage de la clé publique $p_k$ en un texte chiffré $c$.

(e) Dérivation de la clé finale $K$ ($K := KDF(\bar{K}||H(c))$) par l'utilisation de la fonction dérivation de clé $KDF$. Cette fonction prenant en entrée la concaténation de la clé partagée intermédiare $\bar{K}$ et le haché $H(c)$ du chiffré $c$, assure la liaison de la clé partagée qui est sécurisée et au texte chiffré.

(f) Retourner le chiffré $c$ et la clé $K$ précedemment calculés.

**Explication de la Trappe** :
La trappe dans ce code se réfère à la difficulté pour un attaquant de récupérer la clé partagée $K$ sans connaître la clé secrète correspondante à la clé publique $p_k$. La sécurité repose sur les principes suivants :

(a) **Hachage et fonctions de génération :** Les fonctions de hachage $H$ et de génération $G$ assurent que les valeurs dérivées sont imprévisibles et liées de manière complexe aux entrées, rendant difficile toute tentative de prédiction ou de falsification.

(b) **Encapsulation sécurisée :** L'algorithme **CPAKEM.Enc** encapsule le message de manière sécurisée en utilisant des techniques de cryptographie à clé publique. La sécurité de cet algorithme garantit que sans la clé secrète correspondante, il est pratiquement impossible de décrypter $c$.

(c) **Fonction de dérivation de clé :** La fonction $KDF$ génère la clé partagée finale $K$ en combinant des valeurs liées au message encapsulé et au texte chiffré, assurant ainsi que $K$ est sécurisée même si un attaquant parvenait à intercepter $c$.

3. **DecapsulationCCAKEM** : Un algorithme de décapsulation qui a pour rôle de déchiffrer le texte chiffré $c$ et récupérer la clé secrète encapsulée $K$ en utilisant les algorithmes $Kyber.CPAPKE.Enc$ et $Kyber.CPAPKE.Dec$ définis dans le schèma PKE sécurisé IND-CPA.

**Kyber.CCAKEM.Decaps**($sk, c$

(a) Entrée : Texte chiffré $c \in \mathcal{B}^{d_u.k.n/8+d_v.n/8}$

(b) Entrée : Clé secrète $sk \in \mathcal{B}^{24.k.n/8+96}$

$pk := sk + 12.k.n/8$
$h := sk + 24.k.n/8 + 32 \in \mathcal{B}^{32}$

2. En cryptographie, un nonce contribue à éviter les attaques par relecture et empêcher que les anciennes communications soient réutilisées par les cybercriminels.

$z := sk + 24.k.n/8 + 64$
$(\bar{K}', r') := G(m'||h)$
$c' := Kyber.CPAPKE.Enc(pk, m', r')$
si $c == c'$ alors
$K = KDF(\bar{K}'||H(c))$
sinon
retourne $K = KDF(z||H(c))$
fin si retourne $K$.

**Description du code :** : Cet algorithme décrit le déroulement de décapsulation de la clé $K$ comme suit :
**Étapes de l'Algorithme** :

(a) Extraction de la clé publique $pk := sk + 12.k.n/8$ à partir de la clé secrète en prenant une partie spécifique de $s_k$.
(b) Extraction de valeurs supplémentaires à partir de la clé secrète :
- $h := sk + 24.k.n/8 + 32 \in \mathcal{B}^{32}$ définissant l'extraction de la valeur $h$ sur une autre partie de la clé secrète.
- $z := sk + 24.k.n/8 + 64$ qui définit l'extraction de la valeur $z$ d'une partie différente de la clé secrète $sk$.

(c) Génération de la clé partagée intermédiaire $\bar{K}'$ et du nonce $r'$ par l'utilisation de la fonction de hachage $G$ prenant en entrée la concaténation $m'||h$.
(d) Encapsulation hypothétique ($c' := Kyber.CPAPKE.Enc(pk, m', r')$) permettant de génèrer un chiffré $c'$ hypothétique par l'encapsulation de $m'$ et $r'$ avec la clé publique $pk$ vérifiant ainsi si le texte chiffré original $c$ peut être reconstruit.
(e) Vérification de l'intégrité du texte chiffré par les cas suivants :
- Si $c == c$, alors $K = KDF(\bar{K}'||H(c))$ La clé partagée finale $K = KDF(\bar{K}'||H(c))$ est dérivée en utilisant une fonction de dérivation de clé ($KDF$), prenant en entrée $\bar{K}$ et le haché de $c$. Cela indique que le texte chiffré est valide et correspond à $m$ et $r$ ;

- sinon retourne $K = KDF(z||H(c))$, c'est-à-dire, Si le texte chiffré original $c$ ne correspond pas à $c$, une autre clé partagée $K$ est dérivée en utilisant $z$ et le haché de $c$. Cela gère le cas où le texte chiffré a été altéré ou est invalide

(f) Retour de la clé partagée $K$.

## 0.3 Sécurité de Crystals Kyber, Avantages et Inconvénients

### 0.3.1 Sécurité de Crystals Kyber

Le chiffrement CRYSTALS-Kyber est un schéma de chiffrement post-quantique visant à résister aux attaques des futurs ordinateurs quantiques. Ainsi, avec l'avènement de l'informatique quantique où certains ordinateurs quantiques casser certains systèmes comme le RSA avec la factorisation et le logarithme discret, de nombreux protocoles cryptographiques deviendront obsolètes. Kyber, étant un schéma basé sur les réseaux euclidiens, vise à fournir

une sécurité robuste tout en résistant aux attaques de furturs calculateurs quantiques.

Dans les schémas précedents, e problème difficile qui sous-tend la sécurité est le Module-LWE. Il consiste à distinguer les échantillons uniformes $(ai, bi) \leftarrow R_k^q R_q$ des échantillons $(a_i, b_i) \in R_k^q \times R_q$ où $a_i \leftarrow R_k^q$ est uniforme et $b_i = a_i^T \cdot s + e_i$ avec $s \leftarrow B_k^\mu$ commun à tous les échantillons et $e_i \leftarrow B$ nouveau pour chaque échantillon. Plus précisément, pour un algorithme $A$, est défini :

$$Adv_{m,k,\eta}^{MLWE}(A) = |P_r[b' = 1 : A \leftarrow R_q^{m\times k}; (s, e) \leftarrow B_\eta^k \times B_\eta^m; b = As + e; b' \leftarrow (A, b)] - P_r[b' = 1 : A \leftarrow R_q^{m\times k}; b \leftarrow R_q^m; b' \leftarrow A(A, b)]| \text{ [KyAlgoSupDoc]}$$

**Réduction serrée de MLWE dans la ROM**

La réduction serrée de $MLWE$ dans la $ROM$ [3], en combinant ces deux concepts, implique que l'on étudie la sécurité d'un schéma de chiffrement post-quantique basé sur $MLWE$ en supposant que les fonctions de hachage agissent comme des oracles aléatoires. La résistance du schéma aux attaques quantiques et classiques peut être évaluée grâce à cette analyse [KyAlgoSupDoc].

Tout d'abord, le Kyber.CPAPKE est étroitement sécurisé IND-CPA sous l'hypothèse de dureté Module-LWE.

**Théorème 0.3.1** *Prenons l'hypothèse que $XOF$ et $G$ sont des oracles indépendants. Il y a des adversaires $A$ et $B$ dont le temps d'exécution est à peu près le même que celui de $A$, de sorte que $Adv_{Kyber.CPAPKE}^{cpa}(A) \leq \cdot Adv_{k+1,k,\eta}^{MLWE}(B) + Adv_{PRF}^{prf}(C)$.*

On peut facilement prouver ce théorème en observant que, dans l'hypothèse MLWE, la clé publique et le texte chiffré sont supposés être pseudo-aléatoires.

Kyber.CCAKEM. Une transformation de Fujisaki-Okamoto appliquée à Kyber permet d'obtenir CCAKEM.CPAPKE. La sécurité IND-CCA2 de Kyber est démontrée par la déclaration de sécurité concrète suivante.CCAKEM lors de la modélisation des fonctions de hachage $G$ et $H$ comme des oracles aléatoires. On obtient cette conclusion en combinant les limites génériques de génériques avec le théorème précédent (et en optimisant les constantes présentes dans cela).

**Théorème 0.3.2** *Prenons l'hypothèse que $XOF$, $H$ et $G$ sont des oracles incertains. Lorsqu'un adversaire classique $A$ effectue au plus $q_{RO}$ de nombreuses requêtes aux oracles aléatoires $XOF$, $H$ et $G$, il y a des adversaires $B$ et $C$ dont le temps d'exécution est à peu près le même que celui de $A$, ce qui signifie que :*

$$Adv_{Kyber.CCAKEM}^{cca}(A) \leq 2Adv_{k+1,k,\eta}^{MLWE}(B) + Adv_{PRF}^{prf}(C) + 4q_{RO}$$

Il est important de souligner que la mesure de sécurité est rigoureuse. La probabilité d'échec de décryptage de Kyber est connue sous le nom de terme additif négligeable $4q_{RO}$ Kyber.CPAPKE.

3. ROM : La Random Oracle Model est un modèle théorique qui suppose qu'une fonction de hachage (l'oracle) agit comme une boîte noire aléatoire. Dans ce modèle, les fonctions de hachage sont idéalisées et considérées comme des oracles aléatoires.

**Réduction non étanche de MLWE dans le QROM**

La réduction non étanche de MLWE dans le QROM signifie que l'on analyse la sécurité d'un schéma de chiffrement post-quantique basé sur MLWE en supposant que les fonctions de hachage agissent comme des oracles aléatoires quantiques. Cette analyse permet d'évaluer la résistance du schéma face aux attaques quantiques et classiques.

Quant à la sécurité dans le modèle de l'oracle aléatoire quantique ($QROM$), Kyber a été démontré.Dans le $QROM$, $CCAKEM$ est $INDCCA2$ sécurisé, à condition que $Kyber.CPAPKE$ est un $IND-CPA$ fiable. Il est possible d'obtenir une réduction légèrement plus étroite en demandant que le schéma de base Kyber soit utilisé Kyber.CPAPKE est un système pseudo-aléatoire. Il est nécessaire que, pour chaque message $m$, un texte chiffré (obtenu aléatoirement) $(c_1, c_2)$ soit généré pour chaque message $m$ d'une telle façon $(c_1, c_2) \leftarrow Kyber.CPAPKE(pk, m)$ Il est impossible de distinguer un texte chiffré aléatoire de la forme $(Compress_q(u, du)$, $Compress_q(v, dv))$, pour un $(u, v)$ homogène. (La propriété de "disjonction statistique" est également requise, ce qui est facile à remplir pour Kyber.CPAPKE). En effet, la démonstration de la sécurité IND-CPA de Kyber.CPAPKE démontre que Kyber.CPAPKE est étroitement pseudo-aléatoire au niveau de la dureté Module-LWE [KyAlgoSupDoc].

**Théorème 0.3.3** *Prenons l'hypothèse que $XOF$, $H$ et $G$ sont des oracles incertains. Pour chaque adversaire quantique $A$ qui effectue au plus $q_{RO}$ de requêtes aux oracles aléatoires quantiques $XOF$, $H$ et $G$, il y a des adversaires quantiques $B$ et $C$ dont le temps d'exécution est à peu près égal à celui de $A$, de sorte que*

$$Adv^{cca}_{Kyber.CCAKEM}(A) \leq 4q_{RO} \cdot \sqrt{Adv^{MLWE}_{k+1,k,\eta}(B)} + Adv^{prf}_{PRF}(C) + 8q^2_{RO}\delta.$$

La limite de sécurité mentionnée précédemment n'est pas rigoureuse et ne peut donc être utilisée que comme une indication asymptotique de la sécurité CCA de Kyber.Dans le modèle de l'oracle aléatoire quantique, CCAKEM est utilisé.

Après avoir terminé cette partie de la sécurité, nous parlons pour la suite les avantages et les inconvénients du chiffrement Crystals Kyber.

Nous distinguons plusieurs avantages de ce schéma de Crystals Kyber mais aussi des inconvénients.

### 0.3.2 Avantages

Pour les avantages, avant tout nous disons que le chiffrement Crystals Kyber est bénéfique pour la cryptographie surtout qu'il est conçu pour résister aux machines quantiques qui pourront compromettre certains cryptosystèmes comme la factorisation de grands nombres avec RSA prouvé par l'algorithme de Shor et la réduction de la recherche de clé en temps polynomial avec l'algorithme de Grover. Et nous distinguons aussi d'autres avantages :

1. **Efficacité :** En ce qui concerne le temps de calcul et la taille des clés, le cryptosystème présente une efficacité assez élevée par rapport à d'autres systèmes post-quantiques. Les processus de cryptage, de décryptage et de création de clés sont rapides et économiques en termes de ressources informatiques.

2. **Flexibilité et Scalabilité :** Il est possible de configurer CRYSTALS-Kyber avec divers niveaux de sécurité, ce qui permet une adaptation en fonction des besoins spécifiques en matière de sécurité et de performance.
3. **Contributions Théoriques et Pratiques :** La base théorique du système repose sur des problèmes mathématiques étudiés et supposés complexes (comme les Learning With Errors - LWE - et Ring-LWE), ce qui assure sa sécurité. Plusieurs analyses et évaluations ont été effectuées dans le cadre du processus de standardisation de NIST, ce qui a renforcé sa crédibilité et sa confiance.
4. **Compatibilité :** Il est compatible avec les protocoles de sécurité déjà en place et peut être facilement intégré dans différentes applications.

### 0.3.3 Inconvénients

1. **Taille des Clés et des Messages Chiffrés :** Même si les clés publiques, les clés secrètes et les messages chiffrés sont plus petits que celles de nombreux autres candidats post-quantiques, elles sont encore nettement plus grandes que celles des systèmes classiques (RSA ou ECC). Les systèmes qui ont des contraintes de stockage ou de bande passante peuvent rencontrer des difficultés en raison de cela.
2. **Nouvelle Technologie :** CRYSTALS-Kyber étant une technologie relativement récente, elle n'est pas aussi bien adoptée et mature que les systèmes traditionnels. Il pourrait y avoir des vulnérabilités non découvertes ou des défis dans des implémentations spécifiques.
3. **Complexité Algorithmique :** Les concepts avancés de la théorie des réseaux et de la cryptographie utilisés par l'algorithme peuvent rendre son utilisation et son analyse plus complexes pour des développeurs non spécialisés.
4. **Dépendance à des Primitives Non Classiques :** La sécurité est basée sur des hypothèses (comme la complexité du problème LWE) qui, bien que solides et largement acceptées, sont moins connues et moins étudiées que les problèmes classiques tels que la factorisation des grands nombres ou le logarithme discret.

## Conclusion

Brève, Kyber offre une protection contre les attaques IND-CCA2 en utilisant un mécanisme de chiffrement de clé encapsulée (KEM). Elle est sécurisée grâce à la difficulté de résoudre le problème de l'apprentissage avec erreurs (LWE) sur des réseaux de modules. Quelques éléments essentiels : Sécurité : Kyber est l'un des candidats à la cryptographie post-quantique du NIST. Trois ensembles de paramètres distincts sont proposés pour atteindre différents niveaux de sécurité. Vérification de sécurité : L'article actualise l'analyse de sécurité, notamment en fournissant une analyse plus approfondie des attaques au-delà de la difficulté "core-SVP" et en abordant les attaques qui exploitent les échecs de déchiffrement. Les caractéristiques de sécurité de Kyber incluent la capacité à résister aux attaques algébriques, à garantir la confidentialité avant-coup, à faire face aux attaques par canaux latéraux et à faire face aux attaques multi-cibles. En bref, Kyber présente des perspectives prometteuses en matière de cryptographie post-quantique, avec une analyse de sécurité solide et des performances appropriées.

# Bibliographie

## Bibliographie et webographie

### Bibliographie


[FaLa] Fabien Laguillaumie, Adeline Langlois, Damien Stehlé, *Chiffrement avancé à partir du problème Learning With Errors*, Presses universitaires de Perpignan, 2014.

[KyAlgoSupDoc] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, Damien Stehlé, *CRYSTALS-Kyber Algorithm Specifications And Supporting Documentation (version 3.0)*, 2020.

[WaBe] Ward Beullens, Jan-Pieter D'Anvers, Andreas Hülsing, Tanja Lange, Lorenz Panny, Cyprien de Saint Guilhem, Nigel P. Smart, *POST-QUANTUM CRYPTOGRAPHY*, 2021, v2.

[MaVe] Matthieu Vert, *Why IND-CPA implies randomized encryption*, Université Johns Hopkins.

[UdPa] Udara Pathum, *CRYSTALS Kyber pour le chiffrement hybride post-quantique avec Java*, Medium, 2023.
https ://medium.com/@hwupathum/using-crystals-kyber-kem-for-hybrid-encryption-with-java-0ab6c70d41fc

[LyPeOd] Adim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, Advances in Cryptology – EUROCRYPT 2010, volume 6110 of LNCS, pages 1–23. Springer, 2010. http ://www.iacr.org/archive/eurocrypt2010/66320288/66320288. pdf. 7, 12, 34, 35

[CCAKEM] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, Damien Stehlé, Crystals–Kyber : a CCA-secure module-lattice-based KEM, 2018 https ://ieeexplore.ieee.org/stamp/stamp.jsp ?tp=&arnumber=8406610


### Webographie