Projet de recherche :
Side Channel Analysis :
Description des algorithmes :

DILITHIUM :

Dilithium ou « CRYSTALS-Dilithium » est un algorithme de signature basé sur les « lattices problems over mudile lattices », un des principes de Dilithium est que même si un attaquant à accès a un oracle, il ne pourra pas produire la signature d'un message qu'il n'as pas vu, ni de produire une signature différente d'un message qu'il a déjà vu signé.

La sécurité d'une clé est représentée par la taille de la matrice de ses polynômes, plus la matrice est large, plus la clé est considérée comme ayant une forte sécurité, la notation se marque de la suivante :

Une clé utilisant CRYSTAL-DILITHIUM (6,5) aura une matrice de taille 6x5.

Design :

Le design de Dilithum est basé sur la technique proposée par Lyubashevsky nommé « Fiat-Shamir with Aborts » qui se repose sur « rejection sampling to make lattice-based Fiat-Shamir schemes compact and secure »

Clé :

Selon les dires de l'équipe ayant créé l'algorithme, Dilithium possède la plus petite clé publique et la plus petite signature de touts les schemas « lattice-based » qui utilise l'« uniform sampling »

Lors du second round du project NIST, un variant de l'algorithme a été proposé nommé Dilithium-AES, qui utilise un AES-256 en « counter mode » au lieu de « SHAKE » pour étendre la matrice est les vecteurs de masquage, et pour produire les polynômes secrets.

Fonctionnement :

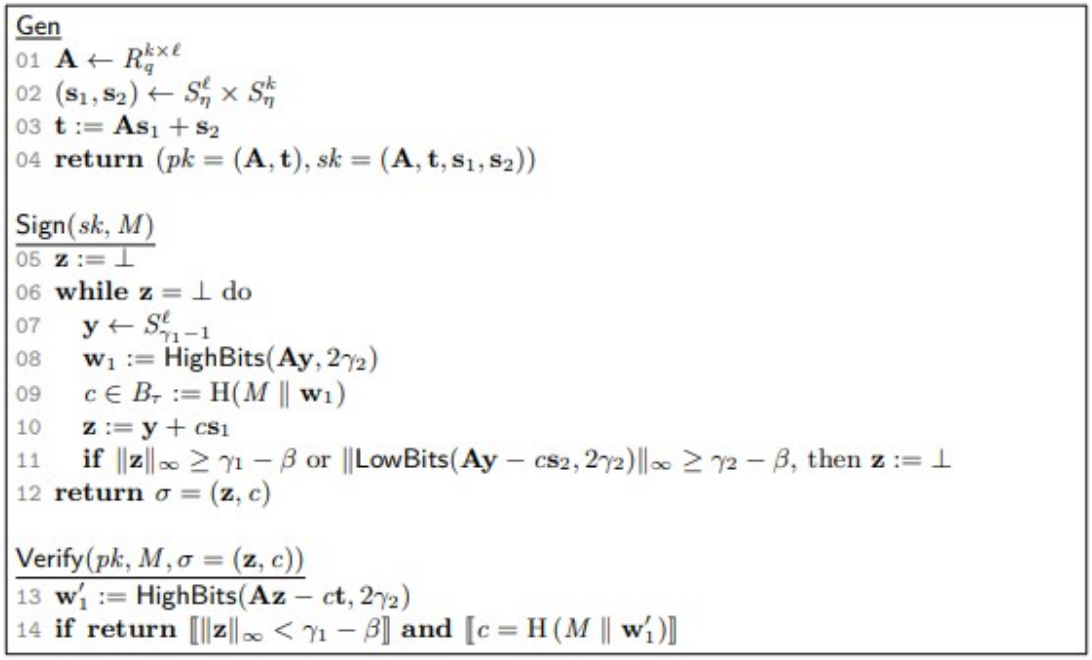

Figure 1: Schéma de la signature sans la compression de la clé secrète

Le fonctionnement de l'algorithme est basée sur l'approche « Fiat-Shamir with Aborts »

Génération de la clé :

L'algorithme de génération de clé génère une matrice A de taille k x l, dont chacune de ses entrées est un polynomial sur l'Anneau Rq = Zq[X] / (X^n +1). Après quoi l'algorithme récupère un échantillon aléatoire de vecteurs clés secrètes s1 et s2. Chaque coefficient de ces vecteurs étant un élément de Rq, de taille minimum n.

Enfin la deuxième partie de la clé publique est calculée avec la formule t = As1 +s2.

Procédure de signature:

L'algorithme de signature génère un masque à partir d'un vecteur de polynômes y avec des plus petits coefficients que r1. Le paramètre r1 étant décidé stratégiquement . Il doit être assez grand pour qu'éventuellement la signature ne révèle pas la clé secrète

Il ne cependant doit être assez petit pour ne pas que la signature soit facilement forgeable. Est ensuite calculé Ay et assigne à w1 le rôle d'être les bits de coefficients du vecteur de « high order »

Chaque coefficient w de Ay peut être écrit dans une solution cannonique tel que w = w1 . 2r2 + w0 ou | w0 | ≤ r2 . W1 étant le vecteur comprends tout les w1.

Le challenge c est ensuite crée en tant que hash du message et de W1.

Vérification :

Le vérifiant compute ensuite w'1 qui sont les bits de « high order » de Az – ct, et accepte si tout les coefficients de z sont plus petit que r1 – B et si *c* est le hash du message et de w'1.

Pour que la vérification soit valide, il suffit de montrer que :

$$\mathsf{HighBits}(\mathbf{Ay}, 2\gamma_2) = \mathsf{HighBits}(\mathbf{Ay} - c\mathbf{s}_2, 2\gamma_2).$$

La raison pour que la vérification soit valide est que nous avons

$$\|\mathsf{LowBits}(\mathbf{Ay} - c\mathbf{s}_2, 2\gamma_2)\|_\infty < \gamma_2 - \beta.$$

Et comme nous savons que les coefficients de cs2 sont plus petits que Beta, additionner cs2 ne causera pas de retenue et de décalages sur les coefficients de « low-order » qui ont au moins une magnitude de r2.

Donc l'équation reste valide, et la signature est donc vérifiée.