Cryptographie post quantique : TP2 - Signature basé sur les réseaux euclidiens

0.1 Paquetages nécessaires

On vous joint ici un ensemble de paquetages sage pouvant vous être utile lors de vos implémentations.

- Distributions Gaussiennes Discrètes sur les réseaux
- Générateur de mauvaises bases
- Hashlib: librairie python non spécifique à sage.

On rappelle également les différentes notations décrites dans ce TP:

- On dénote par $\mathbb{Z}_q := \mathbb{Z}/q\mathbb{Z}$ l'ensemble des entiers modulo q,
- $x \xleftarrow{\$} S$ décrit un tirage uniforme dans l'ensemble fini S.
- $x \leftarrow \mathcal{D}_{S,c,\sigma}$ décrit une distribution gaussienne dans l'ensemble S d'écart-type σ, centrée en c. Par défaut la distribution est centrée en 0.

1 Définitions

(DefSignatureFR) **Exercice 1.** *Définitions de schémas de signature*

1. Donner la définition de schéma de signature.
2. Quels sont les différents types de sécurité d'un schéma de signature ?

2 Génération de trappes pour les réseaux

Avant de passer à l'implémentation d'un schéma de signature, il est important d'implémenter toutes les briques nécessaires, notamment la conception des réseaux avec trappes. L'idée est d'avoir un unique réseau choisi de tel sorte à ce qu'on possède deux bases distinctes: une "bonne" base (que l'on considèrera comme une clé privée) et une "mauvaise" base (que l'on dévoilera en tant que clé publique).
Contrairement au schéma de chiffrement du TP précédent, on ne peut pas utiliser le générateur de réseau difficile fourni par Sage, car celui-ci nous transmet un réseau uniquement avec une mauvaise base, retrouver une bonne base est un problème difficile. (*Il y a d'accord une "compétition" permettant de stess-tester la difficulté de ces problèmes de réseaux: *). L'idée est d'échantillonner parallèlement la mauvaise ET la bonne base. En appliquant un Leftover Hash Lemma, la mauvaise matrice générée paraitra alors complètement aléatoire.

(GadgetMatrixFR) **Exercice 2.** *Matrice Gadget*

Un vecteur gadget $\mathbf{G}$ est un vecteur construit comme $\mathbf{g}^T := \begin{bmatrix} 1 & b & \cdots & b^{k-1} \end{bmatrix}$ (où $\lceil \log q \rceil = k$). Ce vecteur est le vecteur permettant de reconstruire n'importe quel nombre $n$décomposé en base b, tel que $n < b^k$.

On peut étendre la définition à une matrice gadget $\mathbf{G}$. Si on a un vecteur de m décompositions en base b, on étend la définition: $\mathbf{G} := \mathbf{I}_m \otimes \mathbf{g}^T$, où l'opérateur $\otimes$ est le produit de Kronecker (également produit tensoriel sur les).

$$\otimes : \mathbf{A} := (a_{i,j})_{i\in[n],j\in[m]} \in \mathcal{M}_{n,m} \times \mathbf{B}(\in \mathcal{M}_{n',m'}) \longmapsto (a_{i,j}\mathbf{B})_{i,j} \in \mathcal{M}_{nn',mm'}$$

(Tout entier $\mathbf{x}$ se décompose en base binaire $\mathbf{x} \in \Lambda(\mathbf{G})$: tous ces vecteurs $\mathbf{x}$ sont les plus petits éléments avec une norme infinie égale à 1.) Ainsi il est facile d'avoir des solutions de relation $\mathbf{G} \times \mathbf{x} = \mathbf{y}$ on prend simplement la décomposition en base 2 de $\mathbf{y}$ qu'on concatène: cela forme $\mathbf{x}$ (cette décomposition n'est pas forcément unique).

1. Construire une fonction **genGadgetMatrix** retournant la matrice $\mathbf{G}$ pour $q = 2^k$, et pour m vecteurs.

Pourquoi passer par $\Lambda(\mathbf{G})$? car on sait construire une trappe qui est la base $\mathbf{T_G}$ de son réseau dual:

$$\mathbf{T_G} = \mathbf{I}_m \otimes \begin{bmatrix} 2 & & & & & & \\ -1 & 2 & & & & & \\ & -1 & 2 & & & & \\ & & & -1 & \ddots & & \\ & & & & & 2 & \\ & & & & & -1 & 2 \end{bmatrix} \left(\text{si } q = 2^k\right).$$

La matrice à droite du produit est de taille $k \times k$.

2. Etender la fonction précédente afin qu'elle retourne également la trappe-base $\mathbf{T_G}$ associée. Quelle relation a-t'on entre $\mathbf{G}$ et $\mathbf{T_G}$?

3. Construire la fonction de décomposition binaire **decomp** prenant en entrée un entier q et ressortant sa décomposition binaire en base 2.

En réalité, la connaissance de la trappe (ici la base) de $\mathbf{G}$ n'est pas limitée à $q = 2^k$, dans le cas où q est un entier naturel quelconque, on définit la base par:

$$\mathbf{T_G} = \mathbf{I}_m \otimes \begin{bmatrix} 2 & & & & \vdots \\ -1 & 2 & & & \vdots \\ & -1 & \ddots & & \mathbf{decomp}(q)^\top \\ & & \ddots & 2 & \vdots \\ & & & -1 & \vdots \end{bmatrix}$$

où la dernière colonne étant la décomposition binaire de q (bits de poids faible en haut).

4. Généraliser la fonction précédente en **genGadgetTrapdoor** prenant en entrée un entier naturel quelconque $q > 0$, une taille $m > 0$ et ressortant les matrices $\mathbf{G}$ et $\mathbf{T_G}$.

(TrapdoorFR) **Exercice 3.** *Génération de trappes pour les réseaux*

Nous rappelons les différentes étapes à prendre en compte lors de la génération d'une trappe associés à une matrice. L'idée est d'en partant d'une base d'un réseau quelconque, de se ramener au réseau engendré par une matrice gadget $\mathbf{G}$ (avec des coordonnées étant des puissances de 2 i.e. $b = 2$), l'avantage de ce réseau est que les petits élements sont les décompositions naturelles et déterministes des nombres entiers.

1. Construire une fonction **params** permettant d'avoir de manière globale les paramètres, variables et objets qui seront utilisées tout au long de l'exercice (le but est d'avoir une fonction possédant en entrée un modulo q et une taille m, et de mettre à jour toutes les variables liées utilisées).

Nous allons maintenant implémenter l'échantillonage de la trapdoor présentée en [section 5.2].

2. Implémenter une fonction **genLatticeTrapdoor** permettant de construire une matrice cible $\mathbf{A}$ en suivant [Algorithme 1], pour $\mathbf{H} = \mathbf{I}_n$ et en prenant comme distribution $\mathcal{D}$ de $\mathbf{R}$:

$$\mathcal{D} : \{-1, 0, 1\} \to [0, 1] : p(0) = \frac{1}{2} \text{ et } p(\pm 1) = \frac{1}{4}.$$

   Quelle relation a-t'on entre $\mathbf{A}$, la trappe $\mathbf{R}$ et $\mathbf{G}$?

3. **(Facultative)** La fonction précédente permet de générer, en même temps, une matrice pseudo-aléatoire $\mathbf{A}$ et une trappe associée qu'on denotera $\mathbf{R}$. A la différence de l'exercice précédent, ici $\mathbf{R}$ est une trappe mais n'est pas une base du réseau dual.

   Quel lemme de ce même papier permet de construire une base du réseau dual $\Lambda^{\perp}(\mathbf{A})$ à partir de cette trappe $\mathbf{R}$? Rappeler la formule et construire la fonction **genBasisDual** d'entrée $\mathbf{A}$ et $\mathbf{R}$ et ressortant la trappe-base $\mathbf{T_A}$.

4. Dans les schémas cryptographiques efficaces, la clé privée est la trappe $\mathbf{R}$ et non $\mathbf{T_A}$, pourquoi ?

TrapdoorSamplingFR) **Exercice 4.** *Echantillonage avec trappe*

Nous allons maintenant implémenter l'échantillonage gaussien de la [Section 5.4].

1. En utilisant le paquetage de Distributions Gaussiennes Discrètes sur les réseaux, échantillonner un élément $\mathbf{x}$ de $\Lambda^{\perp}(\mathbf{G})$ d'écart-type $\sigma = \sqrt{5}$. (i.e $\mathbf{Gx} = \mathbf{0}$). Vérifier la relation (Utiliser la méthode **.transpose()** sur une de vos matrices pour en sortir la transposée).

2. En utilisant $\mathbf{x}$ et $\mathbf{u}$ un vecteur cible, construisez un élément $\mathbf{z}$ de $\Lambda^{\perp}_{\mathbf{u}}(\mathbf{G})$ (c'est-à-dire que $\mathbf{Gz} = \mathbf{u}$) suivant une loi gaussienne centrée en $\mathbf{decomp}(\mathbf{u})$. Vérifier la relation. Faites attention les décompositions binaires doivent toujours être de même taille $\lceil \log q \rceil$ (il faut remplir avec des 0 les espaces manquants).

3. Généraliser en construisant une fonction **samplePreimageGadget** prenant en entrée un vecteur $\mathbf{u}$ et un écart-type σ.

4. Grâce à la relation entre $\mathbf{A}$, $\mathbf{R}$ et $\mathbf{G}$, et en utilisant **samplePreimageGadget**, construire une préimage $\mathbf{y}$ de $\mathbf{u}$ dans le réseau de $\mathbf{A}$, c'est-à-dire tel que $\mathbf{Ay} = \mathbf{u}$. Vérifiez que la norme de $\mathbf{y}$ est petite.

Comme vu en cours, la difficulté du problème SIS ne réside pas en le fait de trouver une solution quelconque, mais bien de trouver une solution "petite". Trouver une solution quelconque est en effet facile, il faut simplement réaliser un pivot de Gauss et c'est gagner. C'est ce que fait la méthode **.solve_right()**.

5. Généraliser les 3 dernières questions en une unique fonction **samplePreimage** prenant en entrée $\mathbf{A}, \mathbf{R}, \sigma$ et une cible $\mathbf{u}$ ainsi que les différents paramètres globaux.

6. Tester vos implémentations pour des paramètres $m = 32, 48, 64$ (la dimension est le paramètre de sécurité).

(SecureSamplingFR) **(Facultatif) Exercice 5.** *Echantillonage sécurisé*

Si votre construction précédente est valide, alors vous avez généré un algorithme d'échantillonage sur un réseau pseudo-aléatoire. Sans la trappe générée avec le réseau, il est supposé difficile de générer de petits élements. Cependant, pour l'instant, il n'est pas possible de s'assurer de la sécurité de l'échantillonneur et même plus, on sait qu'il n'est pas sécurisé car il fait fuiter des informations sur la trappe $\mathbf{R}$.
Pour cela, on modifie légèrement l'algorithme d'échantillonage comme dans [Algorithm 3], dans une version simplifiée ci-dessous.

Algorithm 1: Echantillonage gaussien sécurisé dans un réseau

Input: une base $\mathbf{A}$, une trappe $\mathbf{R}$ associée, une cible $\mathbf{u} \in \mathbb{Z}_q^n$ et une matrice positive définie Σ.
Output: une préimage $\mathbf{x}$ telle que $\mathbf{Ax} = \mathbf{u}$ où $\mathbf{x}$ suit une loi statistiquement proche de $\mathcal{D}_{\Lambda_{\mathbf{u}}^{\perp}(\mathbf{A}),r\cdot\sqrt{\Sigma}}$
Phase offline:
$\Sigma_{\mathbf{p}} \leftarrow \Sigma - \begin{bmatrix}\mathbf{R}\\ \mathbf{I}\end{bmatrix} (2 + \Sigma_{\mathbf{G}}) \begin{bmatrix}\mathbf{R}^\top & \mathbf{I}\end{bmatrix}$;
assert $\Sigma_{\mathbf{p}} \geqslant 2 \begin{bmatrix}\mathbf{R}\\ \mathbf{I}\end{bmatrix} \begin{bmatrix}\mathbf{R}^\top & \mathbf{I}\end{bmatrix}$
Echantilloner une perturbation $\mathbf{p} \leftarrow \mathcal{D}_{\mathbb{Z}^m,r\sqrt{\Sigma_{\mathbf{p}}}}$;
Phase online:
$\mathbf{v} \leftarrow \mathbf{u} - \mathbf{Ap}$;
$\mathbf{z} \leftarrow$ **samplePreimageGadget**$(\mathbf{v}, r\sqrt{\Sigma_{\mathbf{G}}})$;
Return $\mathbf{x} \leftarrow \mathbf{p} + \begin{bmatrix}\mathbf{R}\\ \mathbf{I}\end{bmatrix} \mathbf{z}$

1. Générer une perturbation $\mathbf{p}$ et calculer $\mathbf{w}$ et $\overline{\mathbf{w}}$ comme dans la phase *offline*.

La phase *online* intègre la cible $\mathbf{u}$ auquel on souhaite la préimage. Cependant, on tiendra ici compte de la perturbation précédente $\mathbf{p}$, ainsi, au lieu de donner une préimage de $\mathbf{u}$, on donnera une préimage *perturbée* par $\mathbf{p}$, assurant la sécurité, c'est-à-dire ne révélant pas d'information sur $\mathbf{R}$.

2. Générer la nouvelle cible perturbée $\mathbf{v}$.

3. Utiliser votre algorithme d'échantillonage précédent pour générer une préimage de $\mathbf{v}$. Puis construire la préimage finale $\mathbf{x}$ de $\mathbf{u}$ ne révélant pas d'informations sur $\mathbf{R}$.

3 Signature GPV

(GPVsignFR) **Exercice 6.** *Signature GPV*

La signature que vous allez construire est issue de l'article: Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. 2008. Trapdoors for hard lattices and new cryptographic constructions. In Proceedings of the fortieth annual ACM symposium on Theory of computing (STOC '08). Association for Computing Machinery, New York, NY, USA, 197–206.
En utilisant les différentes fonctions implémentées précédemment, implémenter le schéma de signature GPV présenté dans la [Section 6.2].

1. La construction basera sa sécurité sur le problème SIS. Construire une fonction **GPVparams** permettant de générer de bons paramètres globaux (tailles) afin d'assurer la sécurité de la construction.

2. Implémenter la fonction **GPVKeyGen** prenant en entrée un paramètre de sécurité λ, et renvoyant une (mauvaise) base $\mathbf{A}$ d'un réseau (définissant la clé de vérification vk) et sa trappe associée $\mathbf{T}$, une bonne base du réseau dual $\Lambda^{\perp}(\mathbf{A})$ (définissant la clé de signature sk). La fonction f_A est définie comme $f_A : \mathbb{Z}_q^m \to \mathbb{Z}_q^n : \mathbf{e} \mapsto \mathbf{Ae}$. Vérifier les conditions à vérifier afin d'assurer la sécurité (elle residera sur la difficulté d'un problème SIS associé de mêmes tailles).

3. La fonction de signature nécessite de signer un haché salé plutôt que le message clair. Pourquoi signer un haché d'un message-salé plutôt que le message clair, plutôt qu'un haché du message ? En utilisant une fonction de hachage, construire une fonction **vectHash** permettant de réécrire la sortie afin qu'elle possède une structure vectorielle adaptée afin d'utiliser les différentes fonctions précédentes.

4. Implémenter la fonction **GPVSign** prenant en entrée un message m et la clé sk, ressortant une signature σ ainsi qu'un sel r.

5. Implémenter la fonction **GPVVerify** prenant en entrée un message m, un sel r, une signature associée σ et la clé vk et ressortant 1 si la signature est valide et 0 sinon. En plus de la bonne relation $f_{\mathbf{A}}(\sigma) = \mathsf{H}(m\|r)$, la signature σ doit valider une autre condition, laquelle ? (idée: il est facile de trouver un $\mathbf{x}$ quelconque tel que $\mathbf{Ax} = \mathbf{0}$).

6. Vérifier l'implémentation de vos fonctions pour $\lambda = 32, 48, 64$.

4 (Facultatif) Extension de la génération de trappe

J. Alwen and C. Peikert. Generating shorter bases for hard random lattices. In STACS, pages 75–86. 2009.

Cash, D., Hofheinz, D., Kiltz, E., Peikert, C. (2010). Bonsai Trees, or How to Delegate a Lattice Basis. In: Gilbert, H. (eds) Advances in Cryptology – EUROCRYPT 2010.

(ExtendTrapdoorFR) **Exercice 7.** *Extension de la génération de trappe*

Nous allons voir différents moyens d'étendre l'utilisation des trappes. Précédemment nous avons vu qu'il était possible de générer une base pseudo-aléatoire $\mathbf{A}$ et une trappe $\mathbf{R}$ en même temps. Cependant, afin de permettre des constructions plus pertinentes, il est possible d'étendre l'utilisation d'une base pour des réseaux différents, dépendant toujours de $\mathbf{A}$.

1. Implémenter l'algorithme déterministe de la [Section 3.3]. Cette algorithme permet d'étendre l'action de la "bonne base" $\mathbf{T_A}$ sur l'entiereté du nouveau réseau et non plus sur les prémières coordonnées sur laquelle elle était associée. L'avantage second en plus de l'extension est que cela peut se faire sans entraver la taille de $\mathbf{R}$ et donc sa qualité. Ici, on utilisera bien $\mathbf{T_A}$ et non pas juste $\mathbf{R}$.

2. Implémenter l'algorithme de la [Section 3.4] permettant de rendre aléatoire la mauvaise base et ne plus la relier directement à la construction de la bonne base.

3. Définir un fonction **gen_lattice_with_trapdoor** ressortant une bonne base et une mauvaise base après avoir eu en entrée les différentes tailles et dimensions souhaitées. (Révérifier bien les conditions nécessaires des différents algorithmes)

4. Construire une fonction d'échantillonage de préimage, comme l'Algortihme 1.