TP 2 RSA & Sac à Dos

Jean-Baptiste LARGERON, Karam Elnasory

December 2024

1 Introduction

Suite à notre précédent TP¨où nous avons vu des méthodes modulaire simple comme le code de César, nous avons vu qu'il y avait de grosses faiblesses sur ces codes. Nous allons voir ici d'autre méthodes pour crypter un message et nous allons pouvoir les comparer à celles du TP précédent.

2 Sommaire des fonctions

Fichier	**fonction**	**Commentaire**
OutilsMath.py	pgcd()	la fonction prend en paramètre deux nombre et vérifie si ces deux nombres sont premier entre eux
OutilsMath.py	euclide_etendu()	Applique la méthode d'euclide-Etendu sur deux nombres. Nous permettant notamment de trouver l'inverse d'un nombre dans un modulo
OutilsMath.py	est_premier()	Test de primalité probabiliste par l'algorithme de Miller-Rabin
OutilsMath.py	premier()	Donne la liste des n premiers nombres premiers en utilisant Miller-Rabin
OutilsMath.py	superCroissante()	fonction prenant en paramètre une liste de nombre et vérifie si celle-ci est super-croissante.
OutilsMath.py	modInv()	fonction alternative permettant de vérifier si deux nombres sont premier entre eux et renvoie l'inverse du prmeier modulo le second
SacDos.py	sacDifficile()	fonction transformant une liste super-croissante en liste pour un sac à dos difficile. Elle prend en paramètre une liste et un compliqueur
SacDos.py	diviseur()	fonction permettant de trouver le faciliteur d'un nombre E. Elle prend en paramètre un nombre et son modulo
SacDos.py	glouton()	permet de crypter / décrypter un message. Prend en paramètre un message et une liste (sac facile ou difficile)
SacDos.py	trouvePQ()	Trouve P et Q tels que N = P * Q, avec P et Q premiers
SacDos.py	traitementFacile()	Permet de transformer un message reçu en message décryptable grâce à un faciliteur passé en paramètre et à un modulo N
RSA.py	generer_premier()	sert a générer un nombre premier de 'bits' bit.
RSA.py	genere_clefs():	sert à générer les clés pour un code RSA
RSA.py	RSA(message):	génère le chiffrement RSA
RSA.py	ChiffreRSA ()	Chiffre un message grâce au code RSA et a ses clefs (couple de paramètre) dont les clés sont publique ou privée

3 Rappel des méthodes

Code Sac à dos

La méthode sac à dos consiste à avoir un message m (convertis en nombre) et une suite de nombre qui vérifie que m est composé de ces nombres.
Ainsi, le sac à dos est dit facile si la suite est dite super-croissante ; c'est à dire que chaque nombre de la suite est strictement supérieur à la somme de tous les nombres qui la précèdent.

Nous utilisons donc cette méthode afin de codé un message. Pour cela nous allons prendre un nombre n supérieur à la somme total des nombres de notre liste et un nombre premier avec ce nombre n que l'on nommera E et qui sera le compliqueur.
Ainsi, nous allons multiplier les nombre de la suite par E afin d'obtenir une nouvelle suite, puis nous donnerons alors le compliqueur et la nouvelle suite pour qu'elle soit décrypter.
Lors du décryptage il suffira alors de trouver D l'inverse de E modulo n afin d'obtenir le simplifieur qui permettra de décrypter le code.

Code RSA

Le code RSA est un code de cryptage assymétrique. C'est à dire que son déchiffrement ne se fait pas de la même manière que son chiffrement.
Ce code repose sur le petit théorème de Fermat.

Le code consiste à calculer un nombre N tel que $N = p \times q$ où p et q sont des nombres premiers.

Puis il faut calculer la fonction $\phi(N) = (p-1)(q-1)$.

On choisit alors E tel que $E < \phi$et $pgcd(E, \phi) = 1$

On peut alors déduire $D = E^{-1} mod \phi$

Ainsi, la clé publique est E et N tandis que la clé privée est D.

On peut alors chiffrer un message M (si celui-ci est inférieur à N) : Le cryptogramme C est alors égale à M^E mod N (M'=C^D mod N si on utilise la clé privée pour résoudre le cryptogramme).

4 descriptif du programme

OutilsMath.py :

pgcd(): la fonction renvoie le plus grand diviseur commun entre deux nombre a et b.

euclide_etendu: la fonction exécute l'algorithme d'Euclide étendu. Cet algorithme nous permet notamment de trouver l'inverse d'un nombre dans un modulo. Nous avons fait le choix de stocker les valeurs de u et v ainsi que de r et q afin de pouvoir retracer le calcul si nécessaire.

superCroissante(): la fonction renvoie True si la suite passée en paramètre est super-croisssante, False sinon. Très utile pour pouvoir exécuter le code du Sac à dos car celui-ci nécessite un "Sac à dos facile", c'est à dire une suite de nombre dont les nombres sont supérieures à la somme des nombres précédents.

modInv() modInv est une fonction qui verifie le pgcd si deux nombres sont premier entre-eux et renvoie l'inverse du premier modulo le second. Elle est une fonction alternative au précédente et nous à servie de fonction de certification du bon fonctionnement du code. Elle est ici utilisée soit pour du gain de temps, soit pour confirmer les tests que nous avons fait sur l'algorithme d'euclide.

SacDos.py:
BobA et BobB: Ce sont les listes qui nous ont servies pour les tests.

SacDifficile(): cette fonction transforme une liste de nombre super-croissante (sac facile) en une liste différente (sac difficile).

faciliteur():Cette fonction permet de retrouver la clé de déchiffrage (ou faciliteur) d'un nombre compliqueur E. Nous avons fait le choix d'utiliser Euclide_Etendu afin d'avoir une trace du calcul si celui-ci s'avère faux.

glouton(): Fonction appliquant l'algorithme du même nom. La fonction renvoie un nombre binaire (sous forme de tableau). Attention, l'algorithme ne marche pas pour un sac difficile, il faut donc avant passage du message, traiter le message avec la foncton faciliteur.

traitementFacile() Permet de transformer un message reçu en message décryptable grâce à un faciliteur passé en paramètre et à un modulo N. Nécessaire pour pouvoir appliquer glouton() par la suite.

RSA.py:

generer_premier(): Nous avions besoin pour le code RSA de nombre premiers. Ainsi cette fonction genere un nombre aléatoire jusqu'à ce que celui-ci remplisse sa condition d'arrêt : que le nombre trouvé est premier.

genere_clefs() Le code RSA nécessite 2 clefs, une privée et une publique. Cette fonction nous permet donc de le faire. (La première étape dans le rappel des méthodes).

RSA(): Cette fonction exécute le code RSA sur un message (de manière ou les clefs sont aléatoires).

ChiffreRSA(): permet de chiffrer (ou déchiffrer si on utilise la clé privée) à l'aide de la clé publique (ou inversement).

5 Jeux d'essais

Pour les jeux d'essais nous avons décidé de créer les fichier d'exercice de notre TP (Ex1.py, Ex2.py, Ex3.py et ExSacDos.py) qui sont des fichiers permettant d'afficher nos jeux d'essais dans un terminal.

6 conclusion

Le Code Sac à dos trouve ses limites, au même titre que les codes du premier TP. Le Code RSA cependant est beaucoup plus difficile à décrypter si l'on ne connaît pas les clefs. De plus, celui-ci est très pratique dans son utilisation. En effet, il est difficile à générer un code RSA mais celui-ci une fois établi ne nécessite que 2 clés relativement "simple" pour coder et décoder.