Tran Florian Groupe 4-2

Freret Tom

**BUT1 - SAE S1-02**

**COMMUNIQUER, EN TOUTE SÉCURITÉ**

**RAPPORT**

Notre code python est organisé de façon simple, une suite de fonctions, nos fonctions sont commentées pour renseigner les entrées, un résumé bref du fonctionnement et les sorties.

Partie 1 – Cryptographie : le chiffrement RSA

Pour commencer nous nous concentrons sur le chiffrage et le déchiffrage d’information avec un système de clé publique et clé privé. Nous avons tous d’abord créer notre fonction ***key\_creation()****.* Dans celle-ci nous avons créé une liste qui contient des nombres premiers grâce à une fonction extérieur ***list\_prime(n)*** qui renvoie une liste de l’ensemble des nombres premiers inférieurs ou égaux à ***n,*** dans celle-ci nous avons pris ***n = 1000***. Donc grâce à cette liste de nombres premiers nous créer deux variables ***p*** et ***q*** qui prennent chacun un entier dans la liste de nombres premiers aléatoirement, on vérifie bien que les deux variables ne sont pas égales. Ensuite nous créons deux autres variables ***n*** à qui nous attribuons la valeur de ***n = p \* q***, et ***Ⴔ*** à qui nous attribuons la valeur de ***Ⴔ(n) = (p - 1)\*(q - 1)*** . Nous allons démontrer cette formule, voici sa démonstration :

Soit p et q, deux nombres premiers, n = p x q, I l’ensemble des x ∈ ℕ tel que x soit inférieur à n et A l’ensemble des x ∈ ℕ tel que x soit inférieur à n et tel que non premier avec n

φ(n) est le cardinale de l’ensemble des nombres naturels inférieur à n, premier avec n donc

φ(n) = Card(I)+Card(A)

Logiquement Card(I) = n-1 et donc φ(n) = (n-1)+Card(A)

Ensuite on cherche Card(A) :

Soit k ∈ (A) par définition k est multiple de p ou de q

On en déduit que k = a x p avec a ∈ ℕ et k < n ou k = b x q avec b ∈ ℕ et k < n

Et donc Card({ k ∈ ℕ | k = a x p avec k ∈ A }) = p – 1

Card({ k ∈ ℕ | k = b x q avec k ∈ A }) = q - 1

On en déduit donc que Card(A) = (p-1) + (q-1)

φ(n) = (n-1) - ((p-1) + (q-1))

φ(n) = pq - 1 - p + 1 - q + 1

φ(n) = pq - p - q +1

φ(n) = (p-1)(q-1) CQFD

Après cela nous avons créé la variable ***pub*** qui prends un valeur un nombre premier parmi la liste des nombres premiers tant que ***pgcd(pub, Ⴔ(n)) == 1*** (***pgcd()*** est une fonction qui prends en entrée deux entiers ***a*** et ***b*** et renvoie ***d*** le plus grand diviseur commun et ***r*** le reste). Nous avons ensuite créé la fonction ***extended\_gcd()*** qui est l’Algorithme d’Euclide étendue prend en entrée 2 entiers ***a*** et ***b*** et qui renvoie trois entiers ***d*** est le plus grand diviseur commun de ***a*** et ***b***, et ***u*** *et* ***v*** le coefficient de Bézout. Et grâce à cette fonction nous créons la variable ***priv*** qui prends en valeur ***u*** de notre fonction ***extended\_gcd()*** avec en entré ***pub*** et ***Ⴔ(n) .*** Enfin La fonction ***key\_creation()*** nous renvoie ***n, pub et priv***.

Avec cela nous avons créé nos clés, alors nous passons à la prochaine fonction ***encryption\_msg()*** qui prends en entrée notre clé publique***(n, pub)*** et un message texte ***msg***qui est convertit en une liste de nombres avec une nouvelle fonction ***convert\_msg()*** qui prends un message textuel et est convertie grâce à la table ASCII, chaque caractères est convertit en un nombre à trois chiffres. Et cette fonction renvoie une liste de nombres groupé de 4 en 4 pour éviter les attaques fréquentielles et ne pas avoir besoin d’un ***n*** trop grand, si le dernier groupe de la liste ne se fini pas par 4 valeurs nous là remplissons avec des 0 grâce à ***zfill()***. Ensuite la fonction ***encryption\_msg ()*** qui prends en entrée ***n,*** notre clé public ***pub, msg,*** et qui va pouvoir manipuler notre message convertit. Le but de cette fonction est de chiffré notre message, alors nous avons créé une liste ***crypted\_msg*** et avec une boucle et la formule du chiffrement de message Me ≡C[n], avec M notre message convertit et C notre clé publique > n, ensuite grâce à une boucle cela nous renvoie la liste ***crypted\_msg*** qui est le message chiffré.

Après cela, nous devons créé la fonction ***decryption\_msg()*** qui prends en entrée ***n,*** notre clé privé ***priv*** et ***msg*** notre message chiffré. Grâce à la formule M≡Cd[n] avec ***d*** l’inverse de ***e*** modulo ***Ⴔ(n)*** on déchiffre C. Ensuite si la clé privée est négatif on utilise l’opposé de notre clé privé et l’opposé du message chiffré grâce au ***extended\_gcd() ,*** pour notre formule car un nombre à la puissance d’une clé privé négatif serait trop petit. Mais nous avons abandonné cette idée de prendre une clé négative car cela créer des problème quelque fois. Ensuite grâce à ***zfill()*** nous remplissons les groupes de 4 à qui il manque des valeurs par des zéro. Une fois fini la fonction nous renvoie notre message déchiffré.

Pour finir nous avons créé une dernier fonction ***reconvert\_msg()*** qui prends en entré notre message déchiffré, on supprime les caractères ASCII égaux à 000 car nous ne les utilisons pas dans les messages et qu’il pourraient causer des erreurs en effet pour créer des messages on ajoute des zéros. Cette fonction nous renvoie le message déchiffré en caractères.

Partie 2 – Code correcteurs

Q 2.1)

Les tables d’addition et de multiplication de F2 sont les suivantes :

|  |  |  |
| --- | --- | --- |
| F2 | F2 | F2 + F2 |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

|  |  |  |
| --- | --- | --- |
| F2 | F2 | F2 x F2 |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

(On remarque que la table d’addition est égale à la table de vérité du connecteur logique OR et celle de la multiplication à AND. Pour pouvoir transcrire ces propriétés en python on va utiliser l’opération modulo 2 car cela va nous renvoyer soit 0 ou 1 ex : 1+1[2] = 0 et 0+1[2] = 1)

Q 2.2)

On a e1 = (1,0,0,0), e2 = (0,1,0,0), e3 = (0,0,1,0), e4 = (0,0,0,1) en additionnant de certains façons ces vecteurs nous devrons retrouver l’ensemble des vecteurs de F42 les voici tous :

(0, 0, 0, 0) = e1 + e1 (1, 0, 0, 0) = e1

(0, 0, 0, 1) = e4 (1, 0, 0, 1) = e1 + e4

(0, 0, 1, 0) = e3 (1, 0, 1, 0) = e1 + e3

(0, 0, 1, 1) = e3 + e4 (1, 0, 1, 1) = e1 + e3 +e4

(0, 1, 0, 0) = e2 (1, 1, 0, 0) = e1 + e2

(0, 1, 0, 1) = e2 + e4 (1, 1, 0, 1) = e1 + e2 + e4

(0, 1, 1, 0) = e2 + e3 (1, 1, 1, 0) = e1 + e2 + e3

(0, 1, 1, 1) = e2 + e3 + e4 (1, 1, 1, 1) = e1 + e2 + e3 + e4

On peut bien représenter tous l’ensemble des vecteurs qui font partie de l’ensemble F42 avec une somme de vecteurs ei.

Q 2.3)

On cherche à déterminer l’ensemble des vecteurs de l’image de l’application linéaire ø (définie dans le sujet). On va donc utiliser les matrices M et v et avec Φ (v) = M · v

Soit M = et v = , on obtient M · v = =

# On en déduit donc que Im(Φ) = { ((a+b+d), (a+c+d), (a), (b+c+d), (b), (c), (d) | (a,b,c,d) ∈ F42 }

Q 2.4)

Soit u, v ∈F42 et u ≠ v, on cherche à montrer que pour tous vecteurs de Im(Φ), d(u, v) >= 3

Tout d’abord à cause des règles d’addition des matrices, on peut déduire que si u ≠ v, Il peut y avoir soit 4 soit 3, soit 2, soit 1 différences entre les éléments de u et v mais aussi qu’une différence équivaut à un élément égal à 1(voir tableau addition Q 2.1). Nous allons donc étudier ces quatre cas avec u + v = (a, b, c, d) ∈F42 :

* Cas avec 4 différences :

S’il y a 4 différences, alors u + v = (1, 1, 1, 1) et donc Φ (u+v) = (1, 1, 1, 1, 1, 1, 1) et w(Φ (u+v)) = 7

* Cas avec 3 différences :

S’il y a 3 différences, il y aura 3 valeurs parmi a, b, c, d qui seront égales à 1 or comme nous avons vu dans la question précédente l’image de l’application Φ = { ((a+b+d), (a+c+d), (a), (b+c+d), (b), (c), (d) | **(**a,b,c,d**)** ∈F42}. On retrouve a, b, c, d seul et donc logiquement w(ø(u+v)) >= 3

* Pour les cas avec 2 et 1 différences, prouvé que w(ø(u+v)) >= 3 n’est pas vraiment flagrant, il faut étudier chaque cas. Alors pour gagner du temps, toute cette démonstration est possible en testant l’ensemble des vecteurs de Im(ø) avec l’ensemble des vecteurs de Im(ø) qui sont différents, la fonction preuve\_Q4 va réaliser cela et renvoie bien que d(u, v) >= 3

On en conclut donc que d(u, v) >= 3 avec u, v de Im(Φ) tel que u ≠ v.

Q 2.5)

On veut prouver que d(u, ũ) < d(v, ũ), on sait que u et ũ ont au minimum six composantes égales.

Ce qui implique que d(u, ũ) est soit égale à 0 ou 1. Mais on sait aussi que u ≠ v et donc par la question précédente que :

d(u, v) >= 3 ⬄ d(u, v) = d(u, ũ) + d(ũ, v) >= 3

Mais on sait que d(u, ũ) = 1 on en déduit que d(ũ, v) >= 2.

On en conclut que d(u, ũ) = 1 < d(v, ũ) <= 2 ⬄ d(u, ũ) < d(v, ũ)

Alice pourra corriger son message si elle est sûr qu’il y est un bit maximum bruité car il suffit de retrouver le vecteur avec la distance égale à 1 en effet on vient de le prouver avec les 2 questions précédentes. Avec l’exemple précèdent u et ũ, On montre que tous les autres vecteurs ont une distance supérieure à la distance du message bruité.

Partie 3 – Communication sécurisée

Dans cette dernière partie, nous allons créer des fonctions pour pouvoir réaliser une simulation d’envoi d’un message avec du bruit. D’abord avec ***convert-binary()*** , cette fonction va prendre en entrée un tableau d’entier qui va être le message crypte donné par la fonction ***encryption\_msg()*** de. Chaque nombre de ce tableau va être converti chiffre par chiffre en un vecteur de composantes binaires, ensuite nous allons appliquer Φ() à ce vecteur et donc obtenir en sortie un tableau de tableau de vecteur de 7 composantes binaires.

Ensuite nous allons appliquer la fonction ***noise()*** qui va simuler un bruitage sur maximum une composante d’un vecteur (avec 1 chance sur 4) à chacun de ces vecteurs pour cela nous allons utiliser la fonction ***simnoise()***.

Apres cette étape il faut enlever le bruit avec ce que nous avons démontré dans la partie précédente. La fonction ***denoise()*** prend en entrée un vecteur de taille de 7 composantes et regarde la distance entre ce vecteur et tous ce de l’ensemble Im(Φ) et si cette distance est égale à 1, cela veut dire que ce même vect a été bruité et donc que le vecteur testé est le vecteur d’origine. La fonction va donc le retourner s’il existe en revanche dans le cas contraire le vecteur d’entrée n’est pas bruité et donc la fonction le retourne. Nous allons donc grâce à cette fonction débruité tous nos vecteurs avec ***denoise\_msg()***.

Et pour finir il faut retranscrire chaque vecteur en un chiffre et reconstruire les nombres du message crypté. Pour cela nous allons utiliser la fonction ***reconvert\_binary()***. Dans celle-ci nous utilisons les propriétés de Φ() qui est que les valeurs de a, b, c, d sont contenus dans le vecteur de 7 composantes. On peut donc recréer chaque vecteur de composantes et donc les chiffres de chaque nombre et les mettre dans un tableau pour pouvoir reprendre notre algorithme RSA.