BUT1 – S.A.E. S1-02
COMMUNIQUER, EN TOUTE SÉCURITÉ

IUT DE NANTES – DÉPARTEMENT D'INFORMATIQUE – 2021

Contact : Johan Leray, Email : johan.leray@univ-nantes.fr

Le but de ce projet, en trois parties, est de comprendre deux outils mathématiques qui entre en jeu lors de communications sécurisées en ligne.

1. La première partie se concentre sur le chiffrage et le déchiffrage d'informations avec un système de clé publique et clé privée. Concernant cette partie, vous aurez un certain nombre de fonctions Python à réaliser pour mettre en œuvre le chiffrage et le déchiffrage d'un message.
2. La seconde partie est une introduction à la notion de code correcteur. On se concentre sur un exemple simple, dont il faudra illustrer les propriétés. Notez que bien qu'il n'y soit pas explicitement question de code en Python, rien ne vous empêche d'en utiliser, cela étant même fortement encouragé.
3. La dernière partie est une mise en œuvre simultanée de ces deux outils afin de simuler comment a lieu une communication sécurisée en ligne.

Vous travaillerez en *binôme*. Vous devrez avoir rendu votre travail avant

le Vendredi 17 décembre 2021, à 23h59

afin de partir en vacances l'esprit tranquille. Pour cela, vous déposerez sur Madoc quatre fichiers :

— un fichier `SAE_S1_02_RSA.py` qui contiendra l'ensemble de votre code python concernant la partie cryptographie ;

— un fichier `SAE_S1_02_Corr.py` qui contiendra l'ensemble de votre code python concernant la partie code correcteur ;

— un fichier `SAE_S1_02_Bilan.py` qui contiendra l'ensemble de votre code python concernant la dernière partie de ce projet ; il pourra faire appel aux aux deux précedents fichiers ;

— un fichier `rapport.pdf` d'au maximum cinq pages qui contiendra l'ensemble des explications, justifications exigées ; ce fichier devra également contenir les noms du binôme.

Vous ne ferez qu'un dépot sur Madoc par binôme. Une partie de la note de cette SAÉ sera une évaluation d'anglais. Pour cette SAÉ, vous disposerez de plusieurs séances pour un total de 12h encadré par un enseignant du département qui pourra vous guider, répondre à vos questions lors de votre travail.

Partie 1 – Cryptographie : le chiffrement RSA

1.1. **Principe du chiffrement RSA**. On considère la situation suivante. Deux personnes, Alice (A) veut communiquer à Bob (B) des informations secrètes par internet. Malheureusement, une troisième personne, Eve (E comme espion ...) veut récupérer ces informations confidentielles à son compte. Malheureusement pour Eve, Alice connaît quelques techniques de chiffrement, notamment le chiffrement RSA (RSA pour Rivest, Shamir et Adleman, les trois inventeurs de ce chiffrement). Le fonctionnement du chiffrement RSA, basé sur des résultats d'arithmétique modulaire, est le suivant.

§ *Asymétrie.* Le chiffrement RSA est asymétrique : il utilise une paire de clés (des nombres entiers) composée d'une clé publique pour chiffrer et d'une clé privée pour déchiffrer des données confidentielles. Les deux clés sont créées par une personne, souvent nommée par convention Alice, qui souhaite que lui soient envoyées des données confidentielles (un message envoyé sera sous la forme d'un nombre entier). Alice rend la clé publique accessible. Cette clé est utilisée par ses correspondants (Bob, etc.) pour chiffrer les données qui lui sont envoyées. La clé privée est quant à elle réservée à Alice, et lui permet de déchiffrer ces données. Le fonctionnement du chiffrement RSA est basé sur la difficulté d'exprimer de grands entiers comme produits d'entiers.

§ *Création des clés.* L'étape de création des clés est à la charge d'Alice.

— Elle choisit deux nombres premiers p et q et calcule le module de chiffrement $n = pq$ (les entiers envoyés par messages devront être inférieurs à n (à noter que la sécurité du chiffrage repose sur le fait que p et q soient grands)

— Elle calcule $\varphi(n) = (p-1)(q-1)$, soit la valeur de l'indicatrice d'Euler en n. **Vous démontrerez cette égalité dans votre rapport** en partant de la définition qui suit : la fonction indicatrice d'Euler associe à tout entier naturel n non nul le cardinal, noté $\varphi(n)$, de l'ensemble des nombres naturels non nuls inférieurs ou égaux à n et premiers avec n).

— Elle cherche un nombre entier e tel que $e \wedge \varphi(n) = 1$ où le symbole $\wedge$ désigne le pgdc (ainsi $\text{pgdc}(a,b) = a \wedge b$ pour tous nombres a et b). Le nombre e est appelé *exposant de chiffrement* et le couple (n,e) constitue la clé publique d'Alice.

— Elle cherche ensuite l'unique nombre entier d inverse de e modulo $\varphi(n)$ et strictement inférieur à $\varphi(n)$ (l'existence de d tient au fait que $e \wedge \varphi(n) = 1$ et on peut le déterminer grâce à l'algorithme d'Euclide étendu pour obtenir l'identité de Bézout (Alice a besoin pour l'obtenir de $\varphi(n) = (p-1)(q-1)$ et donc de p et q qui ne sont connus que d'elle). Le nombre d, appelé exposant de déchiffrement, constitue la clé privée.

§ *Chiffrement du message.* On note M le message que souhaite envoyer Bob. M est un entier naturel strictement inférieur à n. Le message chiffré sera représenté par l'entier naturel C strictement inférieur à n et tel que :

$$M^e \equiv C[n] \ .$$

§ *Déchiffrement du message.* Pour déchiffrer C, on utilise d, car on peut montrer que :

$$M \equiv C^d[n] \ .$$

§ *Justification.* On veut montrer que $M^{ed} \equiv M[n]$. Le nombre d étant l'inverse de e modulo $\varphi(n)$, on en déduit qu'il existe un nombre entier k tel que $ed = 1 + k\varphi(n)$ ou encore $ed = 1 + k(p-$

$1)(q-1)$. On a alors :

$$M^{ed} \equiv M^{1+k(p-1)(q-1)} \ .$$

On envisage à présent deux premiers cas :

— Si M n'est pas un multiple de p alors, d'après le petit théorème de Fermat, on a

$$M^{p-1} \equiv 1[p] \ .$$

En exprimant $M^{1+k(p-1)(q-1)}$ comme $M \times (M^{p-1})^{k(q-1)}$, on obtient que $M^{ed} \equiv M[p]$.

— De manière identique si M n'est pas un multiple de q alors on a

$$M^{q-1} \equiv 1[q] \ .$$

En exprimant $M^{1+k(p-1)(q-1)}$ comme $M \times (M^{q-1})^{k(p-1)}$, on obtient finalement que $M^{ed} \equiv M[q]$.

Ces deux égalités sont en fait vérifiées pour tout entier M. En effet, si M est multiple de p, alors M et toutes ses puissances avec un exposant non nul sont congrues à 0 modulo p (donc $M^{ed} \equiv M[p]$). On obtient de même que si M est multiple de q, alors $M^{ed} \equiv M[q]$. L'entier $M^{ed}-M$ est donc un multiple de p et de q, qui sont des nombres premiers distincts donc premiers entre eux. Le nombre $M^{ed}-M$ est donc également multiple de leur produit $pq = n$ par une conséquence du théorème de Gauss. On a donc bien démontré que $M^{ed} \equiv M[n]$.

1.2. **Attendus pour cette partie**. Vous devez rédiger un ensemble de fonctions permettant la générations des clés publique et privée, le chiffrement d'un message écrit (une chaîne de caractères ne contenant ni caractère accentué, ni majuscule ni ponctuation).

Vous devrez rédiger les fonctions suivantes.

Q. 1.1. Une fonction `list_prime(n)` qui renvoie l'ensemble des nombres premiers inférieurs ou égaux à n.

Q. 1.2. Une fonction `extended_gcd(a,b)` qui prend en entrée 2 entiers a et b et qui retourne trois entiers d, u, v tels que $au + bv = d$.

Q. 1.3. Une fonction `key_creation()`, créant une clé publique et une clé privée `(n,pub,priv)` grâce à deux nombres premiers p et q choisis aléatoirement entre 2 et 1000 en interne dans la fonction.

Q. 1.4. Une fonction `encryption(n,priv,msg)` qui prend en entrée la clé publique `(n,pub)` et un message texte `msg` et qui renvoie le message chiffré (qui pourra prendre la forme d'une liste de nombres).

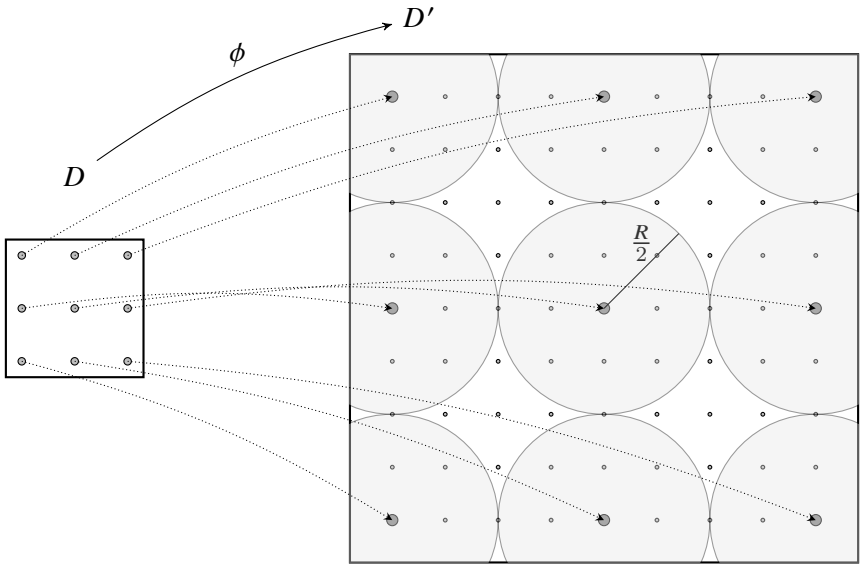
Pour cela, vous pourrez notamment utiliser une fonction `convert_msg(msg)` codée par vos soins qui transforme le message textuel `msg` en une liste de nombres, qui seront ensuite chacun chiffrés. Pour transformer le message textuel, vous pourrez notamment utiliser la table ascii pour traduire chaque caractère en une suite de trois chiffres. (Pour chiffrer le message, on veillera à ne pas le faire caractère par caractère, car un tel chiffrage ne résiste pas à une analyse fréquentielle. Nous aurons peut être l'occasion d'en parler plus tard dans l'année.)

Q. 1.5. Une fonction `decryption(n,pub,msg)` qui prend en entrée la clé privée `(n,priv)` et un message chiffré `msg` et qui renvoie le message décrypté.

Partie 2 – Codes correcteurs

Lors de la communication de l'information en ligne de Bob vers Alice, il se peut que celle-ci soit perturbée, bruitée par un phénomène quelconque. Ainsi, une suite de bits `11011101` au départ peut être reçue comme `11010101` (il y a un bit de différence). Ainsi, comment Alice peut-elle savoir que le message que lui a envoyé Bob est réellement celui qu'il a reçu. Heureusement, Alice sait que l'on peut palier à ce genre de problème, en utilisant des *codes correcteurs*.

L'idée d'un code correcteur est mathématiquement assez simple. On commence par en faire une présentation informelle. On considère D (comme dictionnaire) l'ensemble des mots que l'on veut pouvoir communiquer. Un code correcteur est la donnée d'une application injective $\phi\colon D \to (D', d)$ où D' est un ensemble "plus grand" que D, munie d'une distance $d\colon D' \times D' \to \mathbb{R}$ de telle sorte qu'il existe un réel $R > 1$ tel que, pour tous mots $m_1, m_2 \in D$, alors $d(\phi(m), \phi(m')) \geqslant R$. On représente cela par la figure suivante, où les ensembles D et D' sont représentés par des carrées, les points à l'intérieur de ceux-ci représentant les éléments de ces ensembles.



Dans la suite, on va s'intéresser aux codes correcteurs *linéaires* qui sont appelés ainsi car les ensembles de mots D et D' seront des ensembles de vecteurs (formellement des espaces vectoriels) et l'application ϕ sera codée par une matrice (formellement une application linéaire de D vers D').

2.1. **Une structure mathématique.** On appelle $\mathbb{F}_2$, l'ensemble $\mathbb{Z}/2\mathbb{Z} = \{0, 1\}$. Comme vous l'avez vu lors du semestre, $\mathbb{F}_2$ est ce qu'on appelle *un corps* (il est muni d'une addition et d'une multiplication telle que tous les éléments non nuls soient inversibles pour la multiplication : 1 est son propre inverse ici.)

Q. 2.1. Décrire les tables de l'addition et de multiplication de $\mathbb{F}_2$.

Considérons un entier $n \in \mathbb{N}^*$: on peut alors construire l'ensemble $\mathbb{F}_2^n$ des n-uplets de $\mathbb{F}_2$. Cet ensemble est muni d'une somme : pour $u = (\varepsilon_1, \ldots, \varepsilon_n)$ et $v = (\mu_1, \ldots, \mu_n)$, on a

$$u + v = (\varepsilon_1, \ldots, \varepsilon_n) + (\mu_1, \ldots, \mu_n) \coloneqq (\varepsilon_1 + \mu_1, \ldots, \varepsilon_n + \mu_n)$$

et d'un produit par un scalaire : pour $\lambda \in \mathbb{F}_2$ et $u = (\varepsilon_1, \dots, \varepsilon_n)$, on a

$$\lambda \cdot u = \lambda \cdot (\varepsilon_1, \dots, \varepsilon_n) = (\lambda \cdot \varepsilon_1, \dots \lambda \cdot \varepsilon_n)$$

(on a donc une structure de $\mathbb{F}_2$-espace vectoriel sur $\mathbb{F}_2^n$, structure similaire à celle sur $\mathbb{R}^n$ évoquée en cours).

On suppose à présent que l'ensemble des données que l'on veut transmettre peut s'écrire grâce à des éléments de $\mathbb{F}_2^n$. Dans la suite de cette section, on va considérer le cas où $n = 4$. On suppose donc que l'information que l'on veut transmettre peut se transmettre par des suites de 4 bits : notre premier ensemble D est donc $D = \mathbb{F}_2^4$.

Q. 2.2. On considère les vecteurs $e_1 = (1, 0, 0, 0)$, $e_2 = (0, 1, 0, 0)$, $e_3 = (0, 0, 1, 0)$ et $e_4 = (0, 0, 0, 1)$. Montrer que tout vecteur de $\mathbb{F}_2^4$ peut s'écrire comme une somme de vecteurs e_i. On pourra, par exemple, chercher à résoudre un système linéaire, ou alors exhiber l'ensemble des différents vecteurs obtenus par combinaison des e_i et comparer cet ensemble avec $\mathbb{F}_2^4$.

On va construire notre code correcteur linéaire. On doit donc choisir un nombre $m > 4$ et une application injective $\phi \colon \mathbb{F}_2^4 \to \mathbb{F}_2^m$. Afin d'avoir le code le plus performant possible, ces choix ne sont pas laissés au hasard. On prendra donc $m = 7$ et l'application linéaire $\phi \colon \mathbb{F}_2^4 \to \mathbb{F}_2^7$ sera codé par la matrice

$$M \coloneqq \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(ainsi, pour tout $v \in \mathbb{F}_2^4$, on a $\phi(v) = M \cdot v$)

Q. 2.3. Déterminer l'ensemble des vecteurs de l'image de ϕ. On rappelle que l'image de ϕ est l'ensemble suivant :

$$\mathrm{Im}(\phi) \coloneqq \left\{ y \in \mathbb{F}_2^7 \mid \exists v \in \mathbb{F}_2^4,\ \phi(v) = y \right\}$$

On pourra chercher à résoudre un système linéaire ou alors à exhiber la liste des vecteurs de $\mathrm{Im}(\phi)$.

Ainsi, notre ensemble $\mathrm{Im}(\phi)$ forme notre ensemble D'. Alice décide d'utiliser ce code correcteur pour communiquer avec Bob. Ainsi, pour communiquer un message de 4 bits à Alice, Bob va devoir envoyer 7 bits d'information. Mais reste une question : pourquoi est-ce bien un code correcteur ? Si le message de 7 bits est bruité pendant l'envoi, comment déterminer le message initial ?

2.2. **Correction d'un message bruité.** Dans la suite, nous allons voir que si le message de Bob est altéré d' un bit sur les sept, alors Alice pourra déterminer le message initial. Ainsi, si Bob envoie 1101001 et que Alice reçoit 1111001 alors elle sera capable de retrouver le message initial.

Pour cela, introduisons de nouveau quelques objets mathématiques. On appelle *poids* d'un vecteur $v \in \mathbb{F}_2^7$ le nombre de coefficient non nuls de celui-ci. On note par $\omega \colon \mathbb{F}_2^7 \to \mathbb{N}$ l'application qui à un vecteur associe son poids. Par exemple, le vecteur

$$v = (1, 0, 0, 1, 1, 0, 1)$$

est de poids 4, autrement dit, $\omega(v) = 4$. Grâce à cette notion de poids, on définit de la manière suivante une distance d sur $\mathbb{F}_2^7$. Pour deux vecteurs $u, v \in \mathbb{F}_2^7$, la distance entre u et v est donnée par

$$d(u, v) \coloneqq \omega(u + v) \ .$$

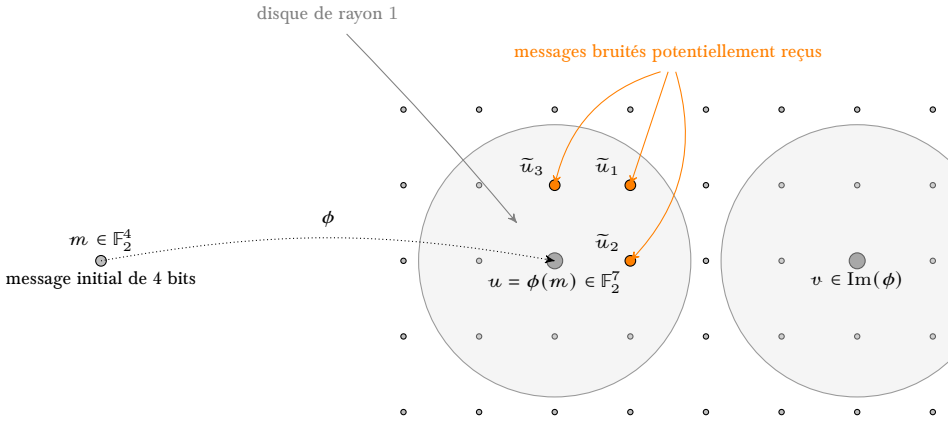
Q. 2.4. Montrer que pour tous vecteurs $u \neq v$ de $\mathrm{Im}(\phi)$, alors

$$d(u, v) \geqslant 3 \ .$$

Supposons à présent que $u \in \mathrm{Im}(\phi) \subset \mathbb{F}_2^7$ soit le vecteur qui représente le message envoyé par Bob, et que $\widetilde{u} \in \mathbb{F}_2^7$ soit le vecteur qui représente le message reçu par Alice (donc u et $\widetilde{u}$ on au minimum six composantes égales), alors, pour tout $v \in \mathrm{Im}(\phi)$, $v \neq u$, on a

$$d(u, \widetilde{u}) < d(v, \widetilde{u}) \ ;$$

(Remarquez que l'inégalité est stricte) autrement dit, le vecteur u est le vecteur le plus proche de $\widetilde{u}$ appartenant à $\mathrm{Im}(\phi)$. On peut illustrer cela par la figure ci-dessous.



Q. 2.5. Montrer le résultat précédent et expliquer que si Alice est sûr que le message de Bob n'a été bruité qu'au maximum sur un bit, alors Alice peut corriger cette erreur.

Remarque

1. En réalité, il existe une méthode très rapide pour trouver le bit où se situe l'erreur, qui repose notamment sur ce choix particulier de matrice M. Mais on ne cherchera pas à déterminer cette méthode ici.
2. Si jamais il y a deux erreurs ou plus dans la transmission de 7 bits, alors notre code ne permet pas de les corriger, ni même de les détecter. Il faudra pour cela améliorer notre code correcteur.

Partie 3 – Communication sécurisée

Pour finir, vous illustrerez comment peuvent être mis en œuvre le chiffrement RSA de la section 1 et le code correcteur linéaire présenté dans la section 2. Cette illustration prendra la forme d'un programme python, qui, partant d'un message textuel que veut envoyer Bob, affichera l'état du message à chaque étape (traduction en nombres, chiffrement, traduction en binaire, envoi avec du bruit, débruitage, déchiffrement, affichage du message final)

Vous devrez notamment utiliser la fonction suivante, qui simule du bruit sur un message de 7 bits que votre code correcteur devrait savoir corriger.

```python
def noise(vect_msg):
    """
    prend un vecteur vect_msg et renvoie ce vecteur potentiellement bruite
    """
    ### on fait une copie du vecteur initial
    vect = vect_msg.copy()
    ### une chance sur quatre de ne pas bruiter le vecteur
    test = np.random.randint(0,4)
    if test>0:
        index = np.random.randint(0,np.size(vect))
        vect[index] = (vect[index] +1)%2
    return vect
```