Travail pratique # 2
Alice, Bob et la cryptographie

Enseignant : Pascal Germain

Date de remise : le lundi 26 octobre 2020
Pondération de la note finale : 12%

1 Objectifs

Ce travail a comme principaux objectifs de valider sa compréhension avec la **décomposition fonctionnelle**, les **chaînes de caractères**, les **listes** et les **fichiers**. Vous constaterez que ce travail demande en premier lieu une **compréhension du code** fourni, et par la suite la réalisation de code permettant de compléter le programme. Nous désirons ainsi vous faire pratiquer une aptitude importante en programmation : la lecture et compréhension de code.

2 Le problème à résoudre

Alice et Bob désirent se communiquer des **messages secrets**, sans que d'autres personnes ne puissent les déchiffrer. Pour y arriver, leurs messages doivent être ***chiffrés*** (ou *encryptés*, mais c'est un anglicisme).

Alice et Bob décident d'utilisent le *chiffrement RSA* (pour *Rivest, Shamir et Adleman*, les inventeurs de cet algorithme). Le chiffrement RSA est l'un des algorithmes les plus utilisés en cryptographie. Dans un système de chiffrement RSA, chaque individu possède une paire de ***clés*** : une ***clé publique***, connue de tous, et une ***clé privée***, connue seulement de l'individu.

Pour envoyer un message secret à quelqu'un, on encrypte notre message à l'aide de sa clé publique. Seul le destinataire sera en mesure de lire le message, car la seule manière de le déchiffrer est d'utiliser sa clé privée ! La figure 1 montre un exemple plus visuel de ces manipulations.

Ce système de chiffrement repose sur le fait qu'il est extrêmement difficile, étant donné une clé publique, de trouver la clé privée associée. Cette difficulté repose sur la **factorisation en produit de deux facteurs premiers** qui, comme vous l'avez probablement constaté dans votre TP1, est très long à réaliser pour de grands nombres.

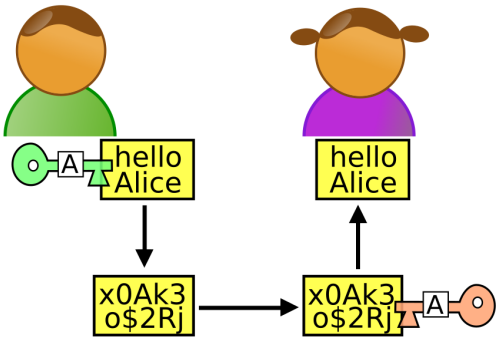

FIG. 1 : Bob désire envoyer le message « hello Alice » de manière privée, à Alice. Il utilise la **clé publique** d'Alice (la clé verte) pour chiffrer son message. Seule Alice peut déchiffrer le message, à l'aide de sa **clé privée** (la clé orange). Source de l'image : Wikipédia.

Nous vous proposons dans ce TP d'implémenter un système simplifié de cryptographie RSA. Pour y arriver, nous vous fournissons une décomposition fonctionnelle dans trois **modules**, où nous avons programmé un sous-ensemble des fonctions. À vous de programmer le code des fonctions manquantes ! La figure 2 montre la décomposition fonctionnelle proposée.

3 Le chiffrement RSA en bref

Tel que décrit plus haut, le chiffrement RSA implique l'utilisation d'une *clé de chiffrement* (clé publique), puis d'une *clé de déchiffrement* (clé privée). Ces clés contiennent trois ingrédients :

- Le *module de chiffrement n* : un entier commun aux deux clés, qui sera utilisé à la fois pour le chiffrement et le déchiffrement.
- L'*exposant de chiffrement e* : un entier (public) utilisé pour le chiffrement. C'est le nombre qui doit être distribué à tous pour leur permettre de chiffrer leurs messages. C'est l'entier que Bob utilise pour chiffrer son message.
- L'*exposant de déchiffrement d* : un entier (privé) utilisé pour le déchiffrement. Cet entier n'est connu que par la personne à qui le message est destiné (Alice).

Ces trois nombres sont obtenus en utilisant deux nombres premiers (en pratique très très grands), avec des opérations mathématiques dont nous vous épargnons les détails. Ces opérations sont déjà programmées pour vous, dans la fonction `generer_paire_de_cles`.

3.1 Le chiffrement

La clé de chiffrement (clé publique) est une **séquence** de deux éléments : le module n et l'exposant de chiffrement e. En Python, on utilisera un *tuple* de deux éléments pour contenir cette clé :

```
ma_cle_publique = (3233, 17)
```

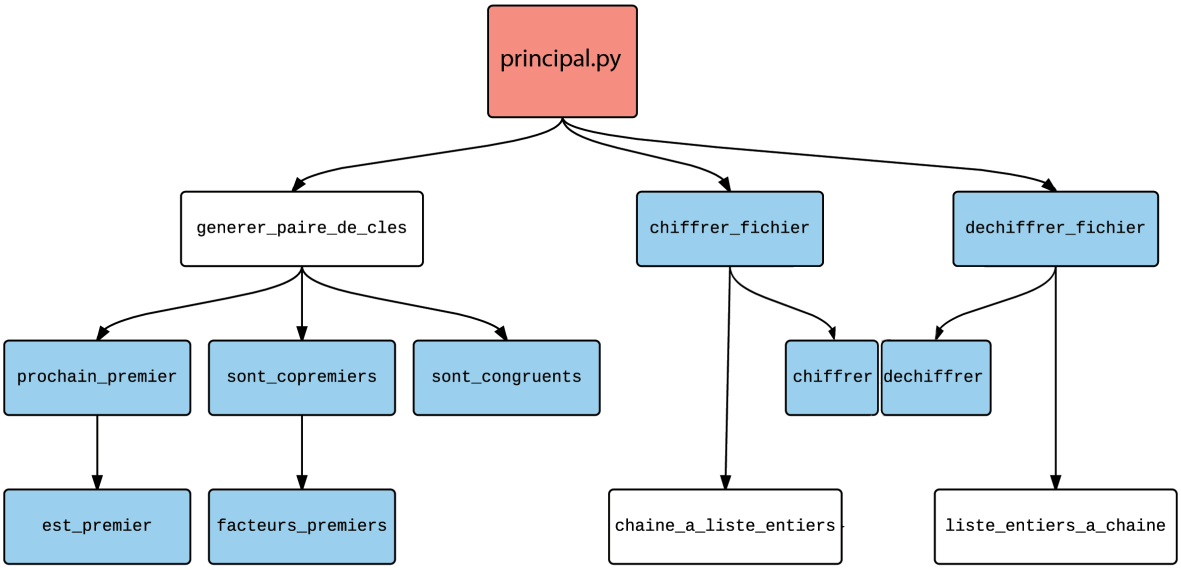

FIG. 2 : Décomposition fonctionnelle du TP2. Une flèche indique une dépendance entre les fonctions. Les fonctions en blanc sont déjà programmées pour vous. Les fonctions en bleu sont à programmer, mais leur définition et leur documentation sont déjà faites. Un exemple de programme utilisant ces fonctions est fourni dans le fichier `principal.py`, mais doit être modifié en suivant les instructions plus bas.

On pourra récupérer les deux entiers de la manière suivante :

```
n = ma_cle_publique[0]  # On récupère le premier élément (3233)
e = ma_cle_publique[1]  # On récupère le second élément (17)
```

Étant donné un message m à chiffrer, obtenons le code c de la manière suivante :

$$c = m^e \mod n\,,$$

où « mod » est l'opération de modulo.

3.2 Le déchiffrement

La clé de déchiffrement (clé privée) est également une **séquence** de deux éléments : le module n et l'exposant de déchiffrement d. En Python, on utilisera un *tuple* de deux éléments pour contenir cette clé :

```
ma_cle_privee = (3233, 2753)
```

Étant donné un code c à déchiffrer, obtenons le message original m de la manière suivante :

$$m = c^d \mod n\,.$$

Supposons que Bob veut chiffrer le message 1234. En appliquant les opérations mathématiques décrites ci-haut,il obtient le message codé suivant :

$$c \;=\; 1234^{17} \mod 3233 \;=\; 2183\,.$$

Le code 2183 peut donc être envoyé à Alice en toute sécurité, car elle seule connaît sa clé privée. Elle retrouve le message original m en utilisant les opérations décrites ci-haut :

$$m \;=\; 2183^{2753} \mod 3233 \;=\; 1234\,.$$

Notons finalement que pour que le chiffrement RSA fonctionne, on ne peut pas choisir n'importe quels entiers d et e ! Ceux-ci doivent avoir des propriétés mathématiques bien définies, dont encore une fois nous vous épargnons les détails. Nous invitons les intéressés à en comprendre davantage à se renseigner sur Wikipédia[1], et à comparer l'algorithme à celui implémenté dans la fonction `generer_paire_de_cles`.

4 Les modules et fonctions fournis

Trois modules de code sont fournis avec ce TP, ainsi qu'un fichier de code `principal.py` utilisant les fonctions de ces modules.

Le fichier `convertisseurs.py` contient deux fonctions : `chaine_a_liste_entiers` et `liste_entiers_a_chaine`. Ces deux fonctions du module `convertisseurs` vous permettent de passer d'une chaîne de caractères à une liste d'entiers (pour ensuite les encrypter), et vice-versa. Consultez la figure 2 pour voir quelles fonctions que vous avez à programmer utiliseront ces deux fonctions fournies. Vous n'avez pas à modifier ce module.

Deux autres fichiers, `rsa.py` et `chiffrement_fichiers.py`, représentent les modules dans lesquels vous devez programmer les fonctions manquantes :

- Le module `rsa` contient une fonction `generer_paire_de_cles` déjà programmée pour vous. Ce module contient aussi une série de tests unitaires pour valider les fonctions du modules que vous devez programmer. Ces tests sont fournis à titre indicatif et sont là pour vous aider à compléter les fonctions demandées sans erreur. Si un test échoue, c'est que votre fonction ne se comporte pas comme elle le devrait.
- Les deux fonctions du module `chiffrement_fichiers` sont à complétées.

La première chose à faire pour attaquer ce TP est de bien comprendre cet énoncé et ce qui vous est fourni. Explorez les fichiers fournis, comprenez l'interaction entre les diverses fonctions.

[1] https://fr.wikipedia.org/wiki/Chiffrement_RSA

5 Les fonctions à programmer

Vous avez 8 fonctions à programmer, dont la définition et la documentation sont déjà présentes. Utilisez cette documentation pour comprendre ce que la fonction doit faire, et si des tests unitaires sont fournis, utilisez-les pour confirmer que votre code est bien fonctionnel.

La fonction `est_premier` a été réalisée en classe et en laboratoire à plusieurs reprises : vous avez le droit de récupérer du code dans le matériel du cours. Ceci dit, programmer cette fonction ne devrait plus avoir de secrets pour vous, et vous devriez y arriver par vous-même sans même regarder les notes.

La fonction `facteurs_premiers` correspond à votre TP1, mais attention : nous ne vous demandons pas d'afficher les facteurs à l'écran, mais bien de les **retourner** dans une **liste**[2] ! Notez également qu'à aucun endroit dans ce TP nous ne vous demandons de poser une question à l'utilisateur, contrairement au TP1.

Noyez qu'au moment de la publication de ce TP, nous n'avons pas encore vu en classe la gestion des fichiers. Cependant, la décomposition modulaire vous permet de programmer les sept fonctions du module `rsa` dans un premier temps, indépendamment des deux fonctions du module `chiffrement_fichiers`. Les tests unitaires du module `rsa` vous permettent aussi de valider la première partie du travail avant de compléter le programme principal.

5.1 Le programme `principal.py`

Le fichier `principal.py` fourni est un exemple d'utilisation des différentes fonctions à programmer dans ce TP. Une fois vos fonctions complétées, nous vous demandons de retirer le code du fichier `principal.py` et d'écrire plutôt du code qui résout le problème suivant.

Nous avons utilisé le système RSA d'une manière alternative : nous avons chiffré le fichier `secret.txt` avec notre clé privée (secrète). De cette manière, toute personne peut déchiffrer le fichier (avec notre clé publique !), mais ceux-ci sont assurés que le message chiffré provient bel et bien de nous, et de personne d'autre. Nous appelons cette opération **signer** une donnée.

Notre clé publique est la suivante :

```
n = 82304707819
e = 17
```

Écrivez dans le fichier `principal.py` le code nécessaire pour déchiffrer le fichier `secret.txt` et l'afficher à l'écran (sans rien demander à l'utilisateur).

[2] Les **listes** sont des objets Python (type `list`) qui se manipulent similairement aux séquences (type `tuple`), mais qui sont en plus *mutables* ; nous verrons en profondeur à la semaine 7 du cours qu'on initialise une liste vide ainsi : `une_liste = []` ; et qu'on ajoute un élément à une liste ainsi : `une_liste.append(element)`.

6 Modalités d'évaluation

Ce travail sera évalué sur 100 points, et la note sera ramenée sur 12. Voici la grille de correction :

Élément	Pondération
Fonction `est_premier`	5 points
Fonction `prochain_premier`	5 points
Fonction `facteurs_premiers`	10 points
Fonction `sont_copremiers`	10 points
Fonction `sont_congruents`	10 point
Fonction `chiffrer`	5 points
Fonction `dechiffrer`	5 points
Fonction `chiffrer_fichier`	15 points
Fonction `dechiffrer_fichier`	15 points
Programme `principal.py` déchiffrant le fichier `secret.txt`	10 points
Respect de la décomposition fonctionnelle demandée	5 points
Qualité du code (noms de variables, style, commentaires, documentation)	5 points

Notez qu'un programme qui n'est pas fonctionnel (qui ne s'exécute pas ou qui plante à l'exécution) pourrait recevoir une note de 0**.**

Votre programme doit être rédigé **à même les fichiers Python fournis**, que vous devez compresser dans une archive Zip (fichier avec extension `.zip`). **Assurez-vous que vous remettez tous les fichiers nécessaires à l'exécution de votre TP**. Nous vous recommandons fortement de créer un **dossier** dans lequel vous mettrez les fichiers relatifs à votre TP, et **rien d'autre** (certains d'entre vous ont la fâcheuse habitude de ne pas organiser leurs fichiers dans des dossiers). Nous ne pourrons pas donner de points à votre travail si vous remettez le mauvais fichier.

7 Remarques

Plagiat : Tel que décrit dans le plan de cours, le plagiat est strictement interdit. Ne partagez pas votre code source à quiconque. Une politique stricte de tolérance zéro est appliquée en tout temps et sous toute circonstance. Tous les cas détectés seront référés à la direction de la faculté. Des logiciels comparant chaque paire de TPs pourraient être utilisés pour détecter les cas de plagiat. Notez que plusieurs exemples de code pour RSA en Python peuvent être trouvés sur Internet. Ne copiez aucun code provenant de ces exemples. S'en inspirer est également découragé : ces exemples n'utilisent pas la même décomposition fonctionnelle que nous, et comprendre/réutiliser ce code sera même plus difficile que de faire le code soi-même !

Retards : Tout travail remis en retard peut être envoyé par courriel à l'enseignant si le portail des cours n'accepte pas la remise. Une pénalité de 10% sera appliquée pour un

travail remis le lendemain de la remise, puis une pénalité de 25% sera attribuée à un TP remis le surlendemain. Une note de 0 sera attribuée pour un plus long retard.

Remises multiples : Il vous est possible de remettre votre TP plusieurs fois sur le portail des cours. La dernière version sera considérée pour la correction.

Respect des normes de programmation : Nous vous demandons de prêter attention au respect des normes de programmation établies pour le langage Python, notamment de nommer vos variables et fonctions en utilisant la convention suivante : `ma_variable`, `fichier_entree`, etc.

Mots-clés interdits : Comme pour le TP1, nous vous interdisons d'utiliser les instructions `break` et `continue`.

8 Ce que vous devez rendre

Vous devez remettre une archive `.zip` d'un **dossier**, contenant les fichiers suivants :

- **`rsa.py`**, **`chiffrement_fichiers.py`** et **`principal.py`** : Les fichiers Python dans lesquels vous avez complété les fonctions et le programme demandé.
- **`convertisseurs.py`** : Le module fourni, nécessaire à l'exécution de vos fonctions dans `chiffrement_fichiers.py`. Vous n'avez pas à modifier ce fichier.
- **`correction.txt`** : Le fichier de correction fourni avec l'énoncé, que vous devez compléter en y indiquant votre nom et votre numéro d'identification.

Vous n'avez pas à remettre les fichiers exemples de fichiers à chiffrer ou déchiffrer : ils sont fournis pour vous aider à réaliser votre TP et les correcteurs utiliseront une version différente.

Cette archive doit être remise via le site Web du cours.

Bon travail !