Pablo Rauzy <pr@up8.edu>
https://pablo.rauzy.name/teaching/is

Introduction à la sécurité

TP 2 : Chiffrement par bloc

Dans ce TP :
- Implémenter un cryptosystème jouet basé sur un réseau de substitution-permutation.
- Comprendre l'utilisation des modes d'opération.

Exercice 0.

Recommandations.

1. Ce TP est à faire en Python.
2. Faites les questions dans l'ordre et pensez à tester votre code.
3. N'hésitez jamais à rajouter de la sortie de debug pour comprendre tout ce qui se passe.

Exercice 1.

Implémentation d'un cryptosystème jouet.

1. Notre cryptosystème, que nous appellerons *ToyCipher*, est un réseau de substitution-permutation à deux tours, avec une clef de 8 bits et une taille de bloc (message) de 4 bits.

   La clef (k) est coupée en deux parties de 4 bits (k_0, k_1) servant de clef de tour.

   Chaque tour consiste en l'ajout de la clef correspondante à l'état (avec un `xor`), puis au passage de l'état à travers une boîte S (on utilise la permutation triviale).

   Pour la boîte S, on peut par exemple réutiliser celle de PRESENT qu'on a vue en cours :

n	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(n)$	c	5	6	b	9	0	a	d	3	e	f	8	4	7	1	2

   → Implémentez la fonction de tour dans une fonction `round`. Pour plus de simplicité, on représentera la clef comme la paire des deux clefs de tour.
2. → Implémentez l'algorithme de chiffrement dans une fonction `enc`.
3. Pour déchiffrer, il faut faire les tours "à l'envers".

   → Implémentez la fonction inverse de `round` dans une fonction `back_round`.
4. → Implémentez l'algorithme de déchiffrement dans une fonction `dec`.
5. Pensez toujours à tester votre code !

   → Vérifiez que les résultats de vos déchiffrements sont cohérents avec les chiffrements, c'est à dire qu'on a toujours l'égalité `m == dec(enc(m, k), k)` pour `m` un message et `k` une clef.

Exercice 2.

Chiffrer plus qu'un bloc.

1. Pour l'instant, votre cryptosystème permet de chiffrer seulement un bloc de 4 bits, c'est à dire un nombre entre 0 et 15. C'est n'est pas franchement utile [1].

   Généralement, on veut chiffrer au moins des messages textuels. Commençons par être capable de chiffrer des lettres, encodée en ASCII, c'est à dire sur un octet.

   → Écrivez une fonction `enc_byte` qui prend un octet et une clef en argument et qui utilise la fonction `enc` pour chiffrer les deux nibbles de cet octet puis retourne l'octet constitué des deux blocs chiffrés concaténés.
2. → Implémentez maintenant la fonction inverse `dec_byte`.
3. En Python, vous pouvez utiliser la fonction `ord` pour récupérer le code ASCII d'un caractère, et la fonction `chr` pour convertir une valeur ASCII en le caractère correspondant.

   → Vérifiez que vous arrivez bien à chiffrer et déchiffrer des caractères de l'alphabet ASCII.

1. Y compris en mettant de côté le fait que c'est un jouet absolument pas sûr à n'utiliser nul part où un besoin, même relatif, de sécurité existe.

4. Maintenant, qu'on sait chiffrer un octet, on veut pouvoir chiffrer n'importe quel fichier.
   → Écrivez une fonction `enc_file` qui prend le nom d'un fichier et une clef en argument, qui ouvre ce fichier en mode binaire et le chiffre octet par octet en utilisant `enc_byte` et écrit le résultat dans le fichier du même nom avec le suffixe `.enc` ajouté.
5. Évidemment, on veut pouvoir déchiffrer.
   → Écrivez la fonction `dec_file` (écrivez le résultat dans le fichier du même nom avec le suffixe `.dec` pour éviter d'écraser vos fichiers existants).
6. → Vérifiez que vos fonctions marchent correctement. Vous pouvez vous amuser à envoyer un fichier chiffrer et la clef secrète à l'un·e de vos camarades pour qu'ille le déchiffre avec son code, pour vérifier que vos implémentations sont compatibles.

Exercice 3.

Jouer avec les modes d'opération.

1. Créez un fichier "test.txt" avec dedans le texte "coucou ? coucou !". Chiffrez le avec la clef `(9,0)`, et observez le résultat.
   → Que remarquez-vous ?
2. → Proposez un type d'attaque contre ce chiffrement.
3. Pour éviter ce genre de problème, on utilise les *modes d'opération*.
   → En vous basant sur les schémas vus cours [2], implémentez au moins l'une des paires de fonctions suivantes (le vecteur d'initialisation peut être généré aléatoirement mais il doit être conservé pour le déchiffrement) :
   — `enc_file_cbc` et `dec_file_cbc`
   — `enc_file_cfb` et `dec_file_cfb`
   — `enc_file_ofb` et `dec_file_ofb`
4. Testez votre code, et vérifiez que le soucis de la première question de cet exercice est corrigé.
   → Quel est l'avantage de la génération aléatoire du vecteur d'initialisation ?

2. Ils sont d'ailleurs tirés directement des pages Wikipédia correspondantes, sur lesquelles vous êtes également encouragé·es à aller faire un petit tour.