Chiffrement par blocs (d'après X mp 2009)

Durée : 2 heures

Notation. Dans tout l'énoncé, $[\![a,b[\![$ désigne l'ensemble des entiers naturels supérieurs ou égaux à a et strictement inférieurs à b.

Lorsqu'on souhaite communiquer des données confidentielles, il convient de *chiffrer* ces données, c'est-à-dire les rendre inintelligibles. Les algorithmes étudiés ici relèvent du chiffrement *symétrique* : une transformation du chiffrement donnée est identifiée par une clé (un entier) qui la désigne et permet également le déchiffrement.

Dans une approche simplifiée du *chiffrement par blocs*, le chiffrement d'un message de taille arbitraire est effectué d'abord en découpant le message en blocs de taille fixée puis en chiffrant chaque bloc. Nous nous limitons ici au chiffrement d'un bloc considéré indépendamment des autres. Dans ce modèle, on se donne un entier $N > 0$, dit *taille* (en pratique N est une puissance de 2). Un bloc (clair ou chiffré) est un entier de $[\![0,N[\![$ et un algorithme de chiffrement est une application de $[\![0,N[\![$ dans $[\![0,N[\![$. Pour permettre le déchiffrement cette application doit être une permutation de $[\![0,N[\![$ (autrement dit une bijection).

Important. Certaines des fonctions demandées sont spécifiées comme renvoyant un tableau. Il faudra utiliser le type *List*.

Partie I. Chiffrement et déchiffrement

On cherche à désigner une application bijective arbitraire de $[\![0,N[\![$ dans lui-même, autrement dit une permutation de $[\![0,N[\![$. On sait qu'il existe N! permutations d'un ensemble de N éléments.
Considérons un entier k (une clé) pris dans $[\![0,N![\![$. On admet que k s'écrit de manière unique sous la forme :

$$k = a_{N-1}(N-1)! + a_{N-2}(N-2)! + \cdots + a_i i! + \cdots + a_2 2! + a_1 1! + a_0$$

où les coefficients vérifient $a_i \in [\![0, i+1[\![$. L'écriture ci-dessus est dite *décomposition sur la base factorielle*. Par exemple, pour $N = 4$ et $k = 17$, on a $k = 2\cdot 3! + 2\cdot 2! + 1\cdot 1! + 0$.

Question 1.

1. À quoi est égal a_0 ?
   Montrer que $k - a_1$ est pair et en déduire une expression simple de a_1 en fonction de k (on pourra noter a mod b le reste de la division euclidienne de a par b).
   De même, exprimer a_2 en fonction de k et a_1 puis plus généralement a_i en fonction de k, a_1, a_2, … , a_{i-1}.
2. En déduire une fonction `decomposerFact(N, k)` qui prend en arguments la taille N et une clé k de $[\![0,N![\![$ et qui renvoie la décomposition de k sur la base factorielle sous la forme d'un tableau $[a_0, a_1, \ldots, a_{N-1}]$.

Une fois k décomposée sur la base factorielle, la permutation σ_k de $[\![0,N[\![$ représentée par k se calcule comme suit. En premier lieu, on considère la séquence $\mathcal{L} = (0,1,\ldots,N-1)$ à N éléments. Cette séquence est modifiée au fur et à mesure que les valeurs prises par la permutation σ_k sont calculées.
La première valeur calculée est $\sigma_k(0)$, égal au $(1+a_{N-1})$-ième élément de $\mathcal{L}$ (c'est-à-dire à a_{N-1}). Une fois $\sigma_k(0)$ calculé, cet entier est retiré de $\mathcal{L}$, qui ne contient plus que $N-1$ entiers.
La seconde valeur calculée est $\sigma_k(1)$, égal au $(1+a_{N-2})$-ième élément de $\mathcal{L}$. Une fois $\sigma_k(1)$ calculé, cet entier est retiré de $\mathcal{L}$.
Le procédé est répété jusqu'au calcul de $\sigma_k(N-1)$ égal à l'unique élément de $\mathcal{L}$ restant.

Par exemple, dans le cas $N = 4$, $k = 17$ on a : $\sigma_{17}(0) = 2$ ($a_3 = 2$), et $\mathcal{L}$ devient $(0,1,3)$. Ensuite $\sigma_{17}(1) = 3$ ($a_2 = 2$) et $\mathcal{L}$ devient $(0,1)$. Ensuite $\sigma_{17}(2) = 1$ ($a_1 = 1$) et pour finir $\sigma_{17}(3) = 0$.

Question 2. Ecrire la fonction `ecrirePermutation(N, k)` qui prend en arguments la taille N et la clé k de $[\![0,N![\![$ et qui renvoie la permutation σ_k représentée par le tableau des $\sigma_k(i)$ dans l'ordre de i croissants.

Question 3. Écrire les fonctions `chiffer(N, k, b)` et `dechiffrer(N, k, b)` qui prennent en arguments la taille N, la clé k et un bloc b. La fonction `chiffrer` renvoie $\sigma_k(b)$ tandis que la fonction `dechiffrer` renvoie l'unique bloc b' tel que $\sigma_k(b') = b$.

Partie II. Réseau de Feistel

Nous prenons ici le parti de fabriquer des permutations *particulières*. Notre motivation est double : (1) réduire la taille des clés (un entier de $[\![0, N![\![$ dans la partie précédente) et (2) effectuer des calculs peu coûteux lors du chiffrement et du déchiffrement.

On commence par fixer la taille à la valeur $N = 2^{64}$. Un bloc b est donc un entier de $[\![0, 2^{64}[\![$. L'ingrédient essentiel du chiffrement est le *réseau de Feistel*. Un réseau de Feistel est une suite de plusieurs opérations, appelées *tours*. Un *tour* est décrit par la figure 1. Sur la figure, l'*entrée* est le bloc $b_i = 2^{32}q_i + r_i$, la *sortie* est $b_{i+1} = 2^{32}q_{i+1} + r_{i+1}$.

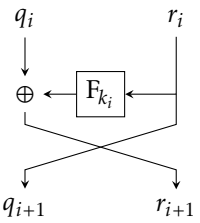

Figure 1 – Un tour de réseau de Feistel

La figure peut aussi se lire comme définissant q_{i+1} égal à r_i et r_{i+1} égal à $q_i \oplus F_{k_i}(r_i)$. Le symbole $\oplus$ désigne ici une opération appelée `xor` et consiste à effectuer un « ou exclusif » sur chacun des bits de la décomposition binaire de x et de y.

$\oplus$	0	1
0	0	1
1	1	0

Figure 2 – Définition du « ou exclusif » de deux bits.

Par exemple, si $x = 13$ et $y = 9$ alors $x \oplus y = 4$ car $x = (1101)_2$ et $y = (1001)_2$ donc $x \oplus y = (0100)_2 = 4$. Cette fonction est associative, commutative et vérifie $(x \oplus y) \oplus y = x$ pour tout couple d'entiers (x, y).
Le symbole F_{k_i} désigne une application sur $[\![0, 2^{32}[\![$, paramétrée par une clé k_i. Par la suite on suppose donnée une fonction Python `f(k, r)` qui calcule $F_k(r)$.

Question 4. Ecrire une fonction `oplus(x,y)` qui prend en arguments deux entiers x et y et qui retourne l'entier $x \oplus y$.

Question 5. Écrire la fonction `feistelTour(k, b)` qui prend en arguments une clé k et un bloc b (k est un certain k_i et b un certain b_i) et renvoie la sortie (notée b_{i+1} ci-dessus) du tour qui utilise la clé k.

Question 6. Écrire la fonction `feistelInverseTour(k, b)` qui réalise l'application inverse de la fonction précédente, c'est-à-dire qui calcule et renvoie b_i en fonction de b_{i+1}.

Question 7. Écrire la fonction `feistel(K, b)` qui prend en entrée le bloc b et renvoie la sortie d'un réseau de Feistel à n tours. Plus précisément, l'entrée b_0 du premier tour est b, puis l'entrée b_i ($i > 0$) d'un tour est la sortie du tour précédent. Enfin, la sortie du réseau est la sortie b_n du dernier tour. Chaque tour utilise une clé différente. Les clés sont fournies (dans l'ordre) par le tableau K de taille n.

Question 8. Ecrire la fonction `feistelInverse(K, b)` qui effectue l'opération inverse de la fonction précédente. Cette opération inverse est le déchiffrement, et l'identité suivante doit être vérifiée pour tout bloc b :

```
feistelInverse(K, feistel(K, b)) == b.
```

♦ ♦
♦